Energy and Performance Prediction of CUDA Applications using Dynamic Regression Models

Shajulin Benedict, Rejitha R.S., and Suja A.Alex
HPCCLoud Research Laboratory
SXCCE, Anna University, India
shajulin,rejitha,suja@sxcce.edu.in
www.sxcce.edu.in/shajulin

ABSTRACT

Many emerging supercomputers and future exa-scale computing machines require accelerator-based GPU computing architectures for boosting their computing performances. CUDA is one of the widely applied GPGPU parallel computing platform for those architectures owing to its better performance for certain scientific applications. However, the emerging rise in the development of CUDA applications from various scientific domains, such as, bioinformatics, HEP, and so forth, has urged the need for tools that identify optimal application parameters and the other GPGPU architecture metrics, including work group size, work item, memory utilization, and so forth. In fact, the tuning process might end up with several executions of various possible code variants.

This paper proposed Dynamic Regression models, namely, Dynamic Random Forests (DynRFM), Dynamic Support Vector Machines (DynSVM), and Dynamic Linear Regression Models (Dyn LRM) for the energy/performance prediction of the code variants of CUDA applications. The prediction was based on the application parameters and the performance metrics of applications, such as, number of instructions, memory issues, and so forth. In order to obtain energy/performance measurements for CUDA applications, EACudaLib (a monitoring library implemented in EnergyAnalyzer tool) was developed. In addition, the proposed Dynamic Regression models were compared to the classical regression models, such as, RFM, SVM, and LRM. The validation results of the proposed dynamic regression models, when tested with the different problem sizes of Nbody and Particle CUDA simulations, manifested the energy/performance prediction improvement of over 50.26 to 61.23 percentages.

CCS Concepts

•**Computing methodologies** → **Model development and analysis;** *Parallel computing methodologies;* •**Software and its engineering** → **Software notations and tools; Software libraries and repositories;** •**Hardware** → *Power and energy;*

ISEC '16, February 18-20, 2016, Goa, India
© 2016 ACM. ISBN 978-1-4503-4018-2/16/02...$15.00
DOI: http://dx.doi.org/10.1145/2856636.2856643

Keywords

Applications; CUDA; Energy; Performance Tuning; Performance Analysis; Tools;

1. INTRODUCTION

GPGPUs are heavily exploited in emerging supercomputers as a section of HPC programmers revealed the improved computing performance of scientific applications [32] by diligently offloading compute intensive CPU codes to GPUs. These programmers utilized computing platforms, such as, CUDA, for offloading CPU codes to GPUs. In addition, they attracted a wide spectrum of scientific programmers for developing CUDA applications. This mission had led to the deployment of tens of thousands of CUDA scientific kernels/applications in recent years.

Although tremendous CUDA applications [28] are available in the HPC market, the performance of those applications are dexterously prone to memory latency issues or energy consumption issues. Excerpts point out that synchronization overhead due to tiling mismatches of iterative stencil codes had led to several research innovations [21] in recent years. In addition, CUDA applications need to be braced with the new releases of CUDA/NVIDIA driver versions which emerge more frequently owing to the change in memory/pipeline structures and number of streaming processors (SMs) of emerging GPGPU architectures.

In general, the performance of CUDA applications is hampered due to several reasons as listed below:

1. the algorithm-architecture mapping problems,
2. display driver issues,
3. global thread scheduling problems,
4. insufficient programmer knowledge, and so forth.

The performance of CUDA applications could be improved if application developers consider memory-hierarchy of CPU-GPUs; similarly, the code conversion procedure (from sequential code to CUDA code) should consider the massive parallelism of CPUs and GPUs.

On the path to exa-scale era, HPC application developers are urged to develop architecture-specific algorithms for obtaining scalable and energy efficient solutions [13, 29], especially in heterogeneous architectures - the focus has shifted decisively. To develop such algorithms, application developers should impose practical-cum-research skills in GPGPUs, CPUs, and CUDA computing platforms in order to avoid the probable performance issues of CUDA applications. In fact, developing those architecture-specific algorithms could be eased with performance tuning tools.

This paper proposed energy / performance prediction mechanisms using Dynamic Regression approaches, such as, Dynamic

Random Forest Models (DynRFM), Dynamic Support Vector Machine (DynSVM), and Dynamic Linear Regresssion Models (Dyn LRM) for CUDA applications. The proposed prediction mechanisms could be applied in energy tuning frameworks [9]. In addition, the proposed energy prediction mechanisms were tested using CUDA applications, namely, Nbody simulation and Particle simulation, on two GPGPU machines i) GEForce GT630 - Ivybridge machine and ii) Quadro 1000M - Sandybridge machine. In succinct, the contributions of this paper are listed as follows:

1. The energy consumption and execution time of the code variants of CUDA applications were predicted using Dynamic Regression models, such as, DynRFM, DynSVM, and Dyn LRM.

2. The proposed energy prediction mechanisms for CUDA applications were compared with classical regression models - LRM, RFM, and SVM.

3. Energy/Performance repository, named EAPerfRepo, containing tens of thousands of performance values that are obtained from the code variants of CUDA applications, were recorded using a no-sql database.

4. EACudaLib, a library that measures the energy/performance values of CUDA applications, was developed in EnergyAnalyzer tool. EnergyAnalyzer tool [31] is a tool that does the energy consumption analysis of HPC applications using RAPL and hardware performance counters.

The rest of the paper is organized as follows. The existing research works on prediction / modeling of HPC applications are described in Section 2. Section 3 explains the energy / performance prediction framework for CUDA based autotuning systems and Section 4 explains the proposed Dynamic models - DynRFM, DynSVM, and DynLRM. The validation of the proposed approach is discussed in Section 5. And, finally, Section 6 presents a few conclusions.

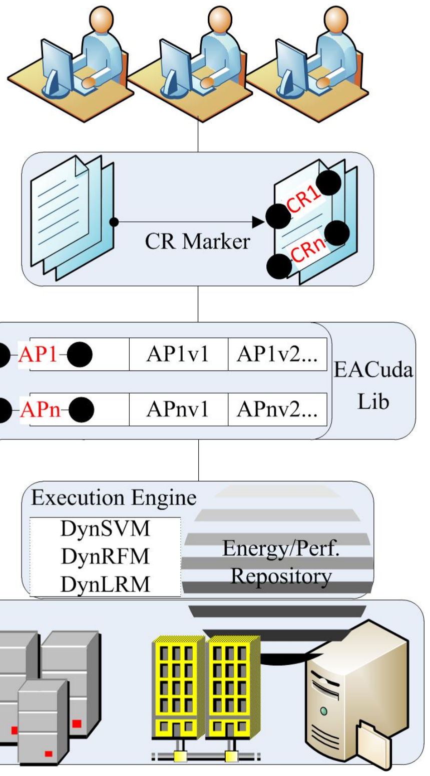


Figure 1: Energy Prediction Framework for CUDA Applications

2. RELATED WORK

HPC applications are abruptly emerging among scientific community owing to the recent advancements of multi-core based supercomputing machines. In past decades, performance prediction of HPC systems and applications had remained as an active research which gave birth to various prediction approaches. For instances, mapping application models to machine profiles [40, 11, 45, 36], analytical models [5, 10, 49], statistical models [41, 6], hybrid analytical and statistical methods [12], historical data [25, 24], time series [19], data mining methods [25, 35], queing theory [46, 39], benchmarking [27, 43], partial program executions [48], simulation [38], skeleton [37]), and machine learning [25] approaches were developed by researchers to predict the execution time of applications.

With the sage of utilizing GPUs for scientific computing, CUDA and the similar computing platforms, such as, OpenCL, and so forth, have taken various strides of performance improvements when compared to the traditional HPC programming approaches. Consequently, several CUDA applications, such as, airspace deconfliction [15], hydro-dynamic application [42], [44], [34] and so forth, have claimed the advancement of GPGPUs in recent years.

Performance evaluation and analysis of CUDA applications, therefore, remained as a crucial step to CUDA application developments [32, 1, 14]. Obviously, CUDA researchers have focused on analyzing the performance issues of CUDA applications. They surfaced their intellects on performance evaluation of CUDA applications in terms of memory issues [22, 16] of codes, code snippets [21], load imbalancing issues of programs, and so forth.

Consequently, CUDA performance analysis frameworks and tools evolved for easing CUDA application developers - GPUPerf [20], campProf [4], cuMapZ [50], and so forth. In addition, a few simulators assisted the programmers for predicting the performance of GPGPUs - for e.g., GPGPUsim supported with AerialVision simulator [3] and Barra simulator [7]. In the meantime, tool developers, performance predictors, compiler designers, and a partial sector of CUDA application developers came up with autotuning proposals and code generation proposals that are specific to applications [18, 23] and compilers [16].

In addition, a few efforts on performance modeling and prediction of CUDA applications were also studied for GPGPUs by researchers. For instances, CPU-GPU workloads were modeled for various CUDA applications by exploring the characteristics of applications in [2]; GPU performance prediction was carried out using data modeling approaches by Michael et al. [8]; additionally, a broad spectrum of researchers got involved in optimizing the sparse matrix vector multiplications [26, 47, 51].

Advancing the traditional performance prediction approaches, recently, researchers have contributed a few power prediction models for CUDA applications. Haifeng et al. [17] proposed power prediction mechanism using fuzzy wavelet ANN approach. Shuaiwen et al. [33] proposed power prediction model using CUDA performance events. Our proposed approach applies dynamic prediction models to predict the energy consumption and the execution time of code variants of CUDA applications. The predictions were based on the performance metrics of CUDA applications.

3. ENERGY/PERFORMANCE PREDICTION OF CUDA APPLICATIONS

This section explains the energy/performance prediction approaches for CUDA applications and the proposed framework for predicting the energy consumption of code variants of CUDA applications. In succinct, the proposed approach dictates that the energy prediction mechanisms could be adopted on energy tuning frameworks.

3.1 Energy/Performance Prediction - A Need

Energy prediction is envisaged as an essential step to autotune CUDA applications albeit performance concerns were considered to be crucial among HPC communities. This is due to the fact that the energy consumption issue is still remaining as a major hurdle to HPC application developers.

In addition, the autotuning process of CUDA applications might end up with several combinations of optimal solutions. Testing all those combinations could be a challenging task, especially when the applications are compute intensive or when the applications are having hefty optimization parameters. The combinations are mostly dependent on the application and machine characteristics of an underlying machine. Therefore, there is a dire need for an energy /performance prediction mechanism during the process of tuning CUDA applications.

In fact, energy tuning process is applied in various levels - i) at compiler level [30], ii) at application level, and iii) at tool level [9]. Our proposed mechanism applies energy / performance prediction mechanism at the tool level.

3.2 Energy/Performance Prediction Framework

In the proposed approach (see Figure 1), we have formulated an energy / performance prediction framework for the code variants of CUDA applications. The steps involved in predicting the energy/performance prediction of the code variants of CUDA applications are described below:

1. Profiling code regions of CUDA applications.
2. Identifying tuning options from CUDA application parameters.
3. Executing a few combinations and framing a model.
4. Predicting the other combinations.

These steps profit the tuning process of CUDA applications (if any) when executed on GPGPU graphic cards. The above mentioned steps are described in detail in the following subsections.

3.2.1 Energy/Performance Profiler

In order to monitor the energy consumption and performance of CUDA applications, a few potential code regions of CUDA applications are manually marked with some specific program statements (CR_n). Precisely, n number of code regions could be marked in a CUDA application for measurements. Each CR_ns are specified as pragmas in the case of C/C++ programs as follows:

```
#pragma CUDA_CRn_begin
....
#pragma CUDA_CRn_end
```

In addition, at the beginning of a main function of the CUDA application, the following profiling function:

```
InitializeEACudalib()
```

is added. This function is responsible for preparing EnergyAnalyzer specific data structures so that the measured performance data would be stored in EAPerfRepo repository.

3.2.2 Application Parameter Extraction & Compilation

Subsequently, CUDA application runtime parameters ($AP\_1..n$) are identified. These application parameters are collected in a list named as TuningList for the runtime optimization purpose. For instance, the particle simulation code of NVIDIA-CUDA-7.5 has parameters, such as, grid size, number of particles, and so forth. The application parameters could be selected such that they are energy/performance optimal for an underlying machine.

In addition, the tuning options for each application parameters, $AP\_1..n$, are noticed for an application in the TuningList. The tuning options are specified in ranges - for instance, the grid size of particle simulation could be varied from $1 \times 1 \times 1$ to $64 \times 64 \times 64$ to $n \times n \times n$. Each combinations of $AP\_1..n$ is denoted as a code variant of CUDA applications. If an application has n number of *APs*, then each combinations of *APs* contributes to frame code variants in our experiments.

Finally, the application is compiled with EACudaLib, a monitoring library developed in the EnergyAnalyzer tool. That is, while executing a CUDA application, the corresponding pragma statements invoke EACudaLib library functions for the measurements. The energy and performance measurements of CUDA applications in EACudaLib are supported with the PAPI hardware counters of CPUs and GPUs. For energy measurements, RAPL counters are used; for measuring the performance events of CPUs, CPU hardware counters are used; and, for measuring the performance events of GPUs, CUPTI driver supported events of CUDA driver are utilized. RAPL counters provide energy measurements in joules for the whole package (socket) and power planes (uncore elements), represented as E_{Pack} and E_{GPU}.

3.2.3 Energy/Performance Modeling

Finding an energy optimal or a performance optimal solution requires searching for the best code variant of CUDA applications. This means that each tuning options (each application parameter based executions) of a CUDA application should be experimented in order to understand the energy consumption and the performance values for those settings. The Execution Engine of our framework is responsible for executing CUDA applications based on the tuning options available in the TuningList.

As executing all the options could be a cumbersome notion, a sequence of available tuning options is split into training and testing datasets by the Execution Engine of our prediction framework (see Figure 1). Thus, instead of executing all tuning variants, only the code variants that are available in the list of training dataset are executed. The performance and energy values for these application parameters are cumulated in an Energy/Performance data repository named as *EAPerfRepo*. The repository is developed using a no-sql database (MongoDB) by extending EAPerfDB of Energy-Analyzer tool.

3.2.4 Energy Prediction Process

In the proposed framework, the performance data that are available on the EAPerfRepo repository are utilized as the training dataset for modeling energy / performance values of CUDA applications. In our approach, both the classical models, such as, RFM, SVM, and LRM and the dynamic models, such as, DynRFM, DynLRM, and DynSVM are applied for predicting the remaining code variants of CUDA applications (that are available in the testing dataset). Thus, prediction based on modeling mechanisms is utilized to support the tuning process of CUDA applications.

A detailed pictorial representation of the proposed energy prediction mechanism for CUDA applications is shown in Figure 1.

4. PREDICTION APPROACHES

Prediction of the code variants of CUDA applications (testing dataset) is devised using dynamic regression mechanisms, such as, Dynamic Random Forest Models (DynRFM), Dynamic Support Vector Machines (DynSVM), and Dynamic Linear Regression Models (DynLRM). This section illustrates the proposed dynamic modeling mechanisms after the modeling basics were explained.

4.1 Regression Models

Although modeling is not a new concept among HPC community, it still remains as a driving factor to quantify the real scenario of performance problems of applications. Both regression and classification approaches are applied by these researchers for predicting the performance concerns of applications.

In general, models require a constant refinement on the pseudo mathematical expression framed from a real world problem. The energy consumption, a dependent variable to the model, is predicted using several independent variables, such as, number of instructions executed, execution time of the CUDA process, problem size of the CUDA kernel, and so forth.

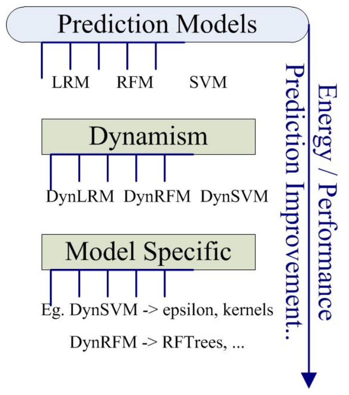


Figure 2: Energy Prediction Regression Models

The pictorial representation of the prediction algorithms that are tested in our framework is shown in Figure 2. We inferred that the energy / performance prediction of the code variants of CUDA applications improved while using dynamism and employing the model specific parameters - the manifestations are reflected in Section 5.

4.2 Random Forest Models

RFM is a tree based modeling technique that reduces the variance of an estimated prediction function. It creates Random Forest Trees (RFTrees) from the training dataset.

RFM Modeling.

RFM modeling undergoes two processes: i) Bagging Process and ii) Ensembling process. During the bagging process, RFTrees are constructed from the training dataset by identifying the independent variables which could remain as a splitting point for creating a branch. During the ensembling process of RFM modeling, the iterative algorithm ensures that the RFTrees are following a sensible approach so that the independent variable is accurately predicted.

RFM Prediction.

RFM prediction process predicts a function $f(x)$ from the testing dataset of the Execution Engine of our framework. The idea behind the RFM prediction mechanism is to average the noisy RFTrees which could consequently reduce the variances of bagging. A detailed explanation about applying RFM in compilers is explained in [30].

4.3 Support Vector Machines

SVM regression is a learning-cum-kernel machine based modeling approach which decides on constructing hyperplanes with higher separation between classes of performance data. Therefore, in SVM regression, a training vector x_i is mapped to sub linear hyperplanes such that a higher margin remains in the higher dimensional space. SVM uses kernel functions and a penalty parameter for modeling the training dataset. If the parameters and kernels are not properly assigned, the SVM models might not produce satisfactory results.

In general, SVM has four basic kernels, namely, linear, polynomial, radial basis function, and sigmoid function. SVM regression undergoes following steps for the effective prediction of independent variables:

1. Transform data: The training data are transformed into an SVM format. For instance, the training data of SVM should be in numbers. If an independent variable has two string options (say, YES or NO), this could be converted to numbers (say, 1 or 0).

2. Adjust kernels: There are four basic kernels and a few associated parameters. For instance, gamma parameter of SVM is not required by linear svm kernel.

3. Train the training dataset using SVM regression model.

4. And, finally, predict the energy consumption or execution time of the testing dataset based on the SVM model.

4.4 Linear Regression Models

Linear regression modeling approach is a commonly used modeling approach. In general, LRM is modeled using least squares mechanism. LRM tries to find the relationship between independent variables and dependent variables. It introduces an error variable ε while finding out the relationship between dependent and independent variables. In fact, LRM assumes that the independent variables x_i of LRM is linear towards the array of dependent variable y_i.

In a simple form, LRM could be mathematically represented as follows:

$$y_i = \beta x_i + \varepsilon_i \qquad (1)$$

where, ε_i is the y_i intercept.

4.5 Dynamic Models - DynRFM, DynSVM, DynLRM

Dynamic models are manifested as a successful modeling approach for predicting time-series data. Time-series data are a set of performance data series (in our context) that are measured over a period of time.

In fact, the energy/performance prediction framework for autotuning purpose, as proposed in our framework, requires continuous execution of code variants over time. Code variants of CUDA applications with respect to different application parameters, therefore, are periodic. For instance, the application parameters of Nbody-CUDA simulation are ranging from $-numbodies$ 256 to $n*256$. Therefore, applying dynamic models would be a perfect fit to the autotuning problem domains - either at compile time or runtime.

Theoretically speaking, dynamic models endeavor to autocorrelate ε_i; whereas, classical regression models consider ε_i as non-parametric which leads to calibration drifts. To counteract the probable drifts, a lagged version of dependent variable could be added as an independent variable while framing regression models. This

Table 1: EAPerfRepo Repository Data used by Prediction Models

CUDA Application	Energy (Machine)	Training Set No. of Entries	Training Set Energy Variation	Training Set Problem Size	Testing Set Entries	Testing Set Energy Variation	Testing Set Problem Size
Nbody	E_{GPU} (ivy)	304	0.3134 to 225.42	256 to 200448	305	0.3183 to 246.7	512 to 206080
Nbody	E_{Pack} (ivy)	299	0.56 to 381.63	256 to 154624	299	0.604 to 373.414	512 to 157696
Nbody	E_{Pack} (sandy)	342	0.7117 to 1773.7	256 to 248320	343	0.66 to 1741.52	512 to 252416
Particle	E_{GPU} (ivy)	1137	0.011 to 14.57	3 to 3513	1138	0.1221 to 17.789	1 to 3508
Particle	E_{Pack} (ivy)	2937	3.364 to 2961.38	1 to 5874	2937	3.21 to 2725.24	2 to 5873
Particle	E_{Pack} (sandy)	2619	6.47 to 3102.93	1 to 5237	2620	6.39 to 4189	2 to 5239

approach could avoid the calibration drifts which mostly occur in the classical regression models, such as, RFM, LRM, and SVM. For instance, in DynRFM, the RFTrees that are formulated in previous iteration of RFM are utilized for framing new RFTrees. As time lagged previous values are applied for modeling, the calibration drift during the regression process is minimized. Thus, in autotuning context, dynamic prediction models would be a perfect fit for the energy prediction of CUDA code variants.

Mathematically, the prediction formula for dynamic regression models in a simple form is given as follows:

$$y_i = \beta x_i + \varepsilon_i + y_{i-1} \tag{2}$$

In succinct, DynLRM, DynRFM and DynSVM fundamentally include the previous value of the independent variable in order to avoid the calibration drift.

5. EXPERIMENTAL RESULTS

This section explains the chosen experimental setup; reveals the prediction results of the code variants of CUDA applications; compares the prediction results of classical prediction models, such as, RFM, LRM, and SVM; explores the importance of Dynamic prediction models, such as, DynRFM, DynLRM, and DynSVM; investigates into the betterment of prediction results by selecting the apt model-specific parameter values.

5.1 Experimental Setup

Experiments were conducted on sandybridge (HPProliantElite 8560w) and ivybridge processor machines (Toshiba-Satellite-P850) of the HPCCLoud Research laboratory of our premise. These machines have Quadro 1000M and GEForce GT630 graphic cards with compute capability of CUDA 2.1x; these machines have single-GPUs. The GPGPUs of both the machines have 96 cuda cores. These GPGPU cards were loaded with NVIDIA driver versions, namely, 355.11 for Quadro1000M and 352 for GEForce (GT630). Both the devices are loaded with the same CUDA display driver namely CUDA-7.5.

5.1.1 CUDA Applications

The CUDA applications (v.7.5 of NVIDIA) that are considered for validating our proposed dynamic regression modeling approaches are described below:

Nbody Simulation.

Nbody simulation is widely applied in physics and astronomy in order to study the influence of particles from their neighbours. In CUDA-7.5, Nbody simulation is implemented to study the influence of gravitational forces between galaxies or stars. The Nbody simulation, by default, works with 16384 number of particle bodies (*numbodies*) and shows that the Nbody simulation performs well - the particle interactions are calculated in a brute force manner. The default option of setting the number of interacting bodies could be overridden by setting command line argument $-numbodies$ to specify the number of bodies involved in the Nbody simulation. The $-numbodies$ parameter of Nbody simulation could be varied in the order of 256.

Each GPU graphic cards might be comfortable with a set of Application Parameter (AP) setting. Finding an optimal application parameter setting based on performance/energy values, therefore, requires an untiring effort to walk through the combinations.

Particle Simulation.

A particle simulation technique is widely applied in science for simulating physical systems and for studying the interactions between particles. The particle simulation implemented in CUDA-7.5 does parallel simulation of particles and their interactions that are residing within a grid structure.

Comparing Nbody simulation, the Particle simulation is implemented such that N particles may not interact with all the other available particles. This means that a few parallel computations are vividly avoided in Particle simulation as those particles do not show any significant interacting forces among the other neighboring particles.

5.1.2 Prediction Setup

In order to predict the performance / energy consumption of the code variants of CUDA applications, the Execution Engine of the proposed energy prediction framework for CUDA applications splits the possible combinations of CUDA code variants as training and testing datasets. This split is done using an equi-distant sampling approach of R programming language. This means that 50 percentages of training and 50 percentages of testing datasets are framed during the prediction process. Later, the training dataset is executed on the underlying CPU-GPGPU machines.

Modeling Variables.

For modeling and predicting the CUDA code variants, the following variables var_i are utilized.

1. var_1: TOT_INS: Total number of instructions.
2. var_2: ExecutionTime: Execution time of code regions of CUDA applications.
3. var_3: E_{Pack}: Energy consumption of package in Joules. E_{Pack} includes the energy consumption of a CPU-GPU socket.
4. var_4: E_{GPU}: Energy consumption of uncore elements, including graphic cards.
5. var_5: L3_TCM: Level 3 cache misses of CPUs.

6. var_6: L1_DCM: Level 1 data cache misses of CPUs.
7. var_7: BR_UCN: Unconditional branch instructions of CPUs.
8. var_8: ProblemSize: Problem size of CUDA code variants.
9. var_9: CUDA_SM_ElapsedTime: Time spent at Streaming Multiprocessors (SM). This option is supported in CUDA 6.5 display driver version.

When E_{Pack} is predicted for CUDA code variants, all the other variables are subjected to act as independent variables. Similarly, when the $ExecutionTime$ variable acts as a dependent variable, this variable would not be included in the list of independent variables of the modeling / prediction process.

To validate the proposed models, R^2 is utilized - R^2 is a measure of goodness of fit for regression models. The R^2 values range between 0 and 1 - 1 assuming a good fit and 0 otherwise. However, the meaning of R^2 values differ from case to case (modeling problem domains).

EAPerfRepo Repository Contents.

As discussed in Section 3, a few portions of CUDA code variants are split as training dataset in the Execution Engine of the prediction framework. It is to be noticed that EACudaLib, a library developed to measure performance values of CUDA applications, measures both the package energy (E_{pack}) and uncore energy (E_{GPU}) for the ivybridge machine; whereas, it measures only E_{pack} in Joules for the sandybridge machine. The energy consumption could be measured only in the recent hardware. In addition, a few other performance values are measured for both the machines. Subsequently, the performance data, including energy consumption values, are stored in the EAPerfRepo repository.

For predicting the energy consumption and execution time of CUDA applications under consideration, such as, Nbody and Particle, the training and testing dataset that are utilized by RFM, LRM, SVM, DynRFM, DynLRM, and DynSVM modeling mechanisms are listed in Table 1.

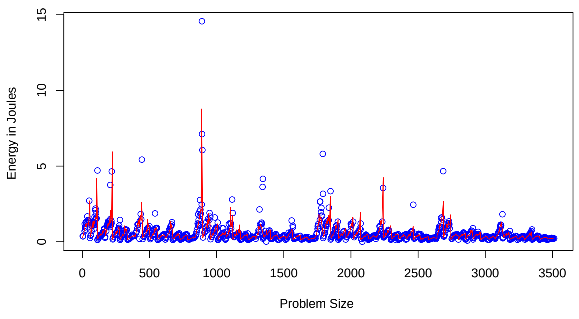
As shown in Table 1, 50 percentages of the data are utilized as training dataset and the other data are utilized as testing dataset. The energy variations and the problem sizes were also listed in the Table 1. The energy variation column of the table dictates the minimal and the maximal energy consumption values for CUDA code variants in Joules.

Nbody simulation considered $-numbodies$ application parameter from 256 to 248320 for sandybridge machine. These values correspond to the problem size of Nbody application. Particle simulation considered two application parameters (AP1 and AP2), namely, particle size and grid size, for representing the problem size. AP1 represented particle size with a minimum value of 500 and AP2 represented grid size of the Particle simulation with the maximum value of 64x64x64. Therefore, AP1=500 and AP2=1x1x1 was represented as ProblemSize= 1 and AP1=600 and AP2=1x1x1 is represented as ProblemSize=2, and so forth, in our experiments.

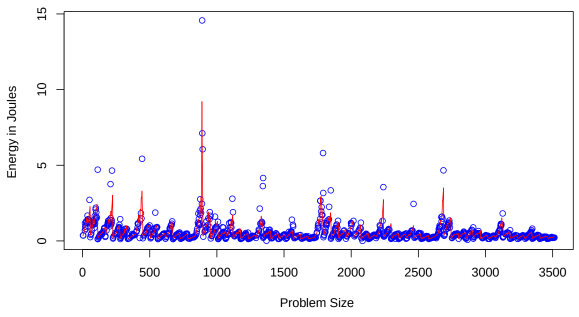
5.2 Prediction Results - CUDA applications

Based on the training and testing datasets (mentioned in Table 1 for Nbody and Particle simulations), predictions were carried out using various models, such as, DynRFM, DynSVM, DynLRM, RFM, SVM, and LRM approaches.

Figures 3 to 8 compare the actual (blue) and predicted (red) energy consumption values of testing dataset of CUDA applications under consideration when experimented with six models. Figures 3

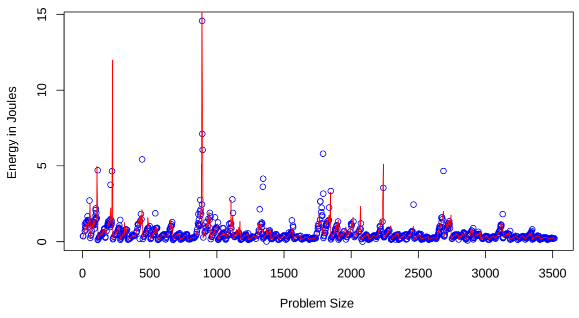


(a) DynRFM.

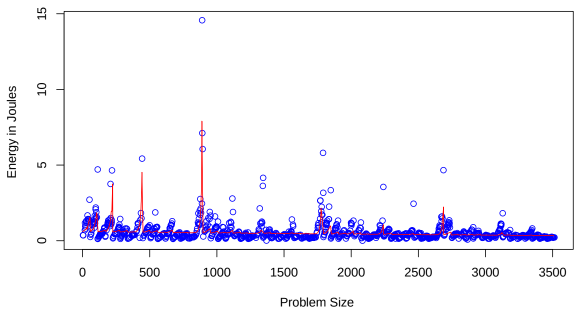


(b) RFM.

Figure 3: Predicted vs. Observed E_{GPU} in Joules for CUDA Particle (v.7.5) simulation - DynRFM and RFM.

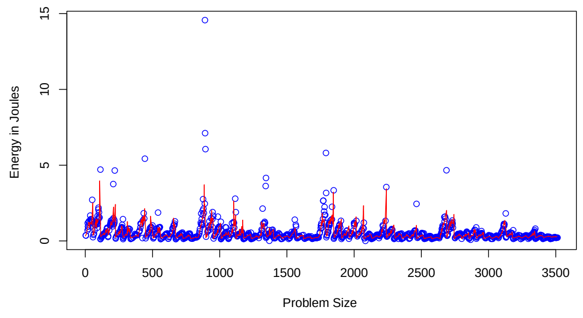


(a) DynLRM.

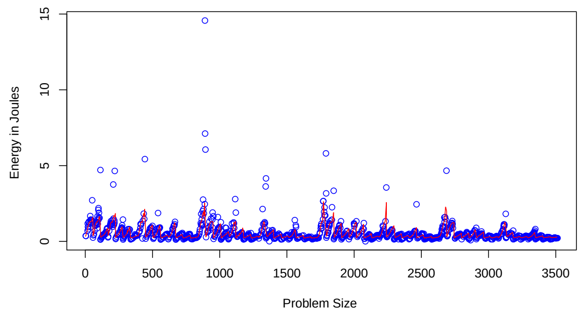


(b) LRM.

Figure 4: Predicted vs. Observed E_{GPU} in Joules for CUDA Particle (v.7.5) simulation - DynLRM and LRM.

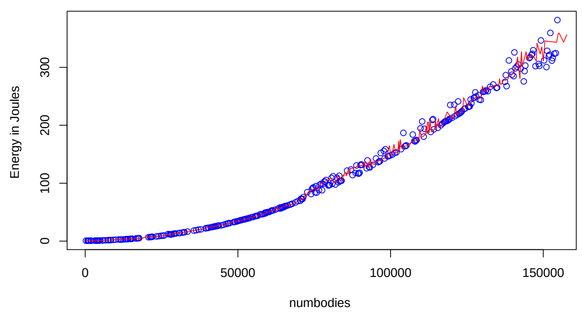


(a) DynSVM.

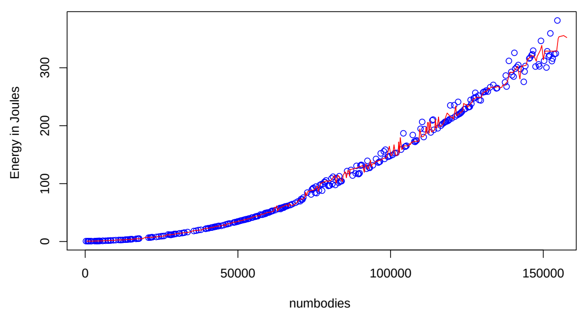


(b) SVM.

Figure 5: Predicted vs. Observed E_{GPU} in Joules for CUDA Particle (v.7.5) simulation - DynSVM and SVM.



(a) DynRFM.



(b) RFM.

Figure 6: Predicted vs. Observed E_{Pack} in Joules for CUDA Nbody (v.7.5) Simulation - DynRFM and RFM.

to 5 show the energy E_{GPU} predicted values in Joules for uncore elements of the socket when experimented with GEForce (GT630)-Ivybridge machine. From these figures, we could observe the following:

1. the energy consumption values of Particle simulation increased with respect to the particle size and grid size of the application. The non-exponential behavior of the energy consumption patterns of the particle simulation is due to the fact that the grid size of the particle simulation increased from 1x1x1 to 64x64x64 for every particle size (see Figures 3 to 5).

2. the energy prediction values due to dynamic models are comparatively better than the classical prediction approaches. For instances, the predicted energy values of particle simulation almost inclines with the observed values of the testing dataset. Similar was the case when comparing DynRFM towards RFM and DynSVM towards SVM.

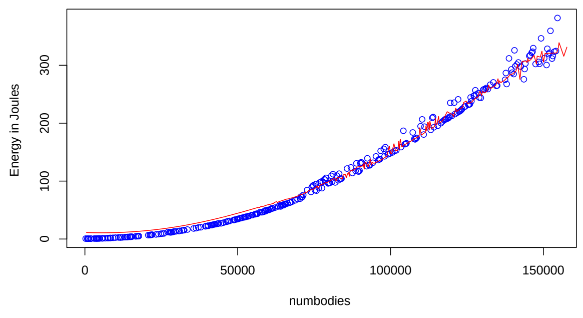
3. However, DynRFM and DynSVM did not perform well when compared to DynLRM. This was due to the inefficient selection of the prediction algorithm specific parameters. A detailed description about the energy and performance prediction improvements while varying the parameters of prediction algorithms, such as, DynRFM and DynSVM is bestowed in Section 5.5.

Similarly, the E_{Pack} values of Nbody simulation was tested on GEForce (GT630)-Ivybridge machine. Contrary to the Particle simulations, the energy consumption values of Nbody simulation were gradually increasing with respect to the number of bodies (-numbodies). However, the energy prediction findings observed for the prediction approaches relied the same - Dynamic models predicted better than the classical models.

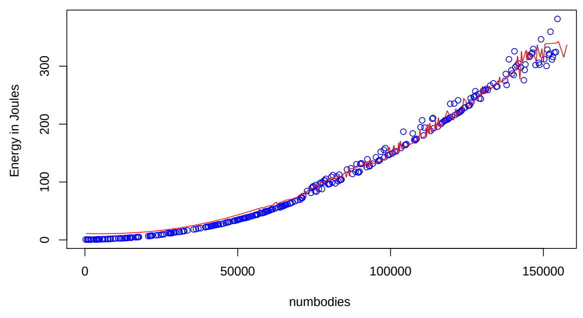
It was interesting to understand the energy and the performance behavior of the applications when tested on different GPGPUs. On experimenting Nbody simulation on the Sandybridge machine with Quadro1000M GPGPU card, the following key points were observed (see Figures 9a and 9b):

1. the E_{Pack} values observed in Ivybridge machine for Nbody simulation were far better than the values obtained at the sandybridge machine. For instance, E_{Pack} values for Nbody simulation at numbodies=51200 showed 131.24 Joules in Sandybridge machine and 36.25 Joules in Ivybridge machine. Thus, the energy consumption values for CUDA applications on ivybridge processor machine with GEForce(GT630) was three times (in avg.) better than the Sandybridge machine.

2. there were a few abrupt increases in the E_{Pack} values when experimented with different problem sizes of Nbody simulation (see Figure 9). This happened on both the machines owing to the delay in memory accesses. However, the abrupt increase in the E_{Pack} values could be easily identified on the Sandybridge machine. For instances, the E_{Pack} values of the Sandybridge machine for n=148 (numbodies=37888) was 36.42 Joules and n=151 (numbodies=38656) was 73.65 Joules. The other abrupt change in the E_{Pack} values was noticed for n=457 (582.45 Joules) and n=465 (792.4 Joules); similarly, on the Ivybridge machine, the E_{Pack} values increased from 81.31 Joules (numbodies = 73984) to 89.85 Joules (numbodies = 74240). However, the execution time of the application did not show such an abrupt change (either increase or decrease) in the values.

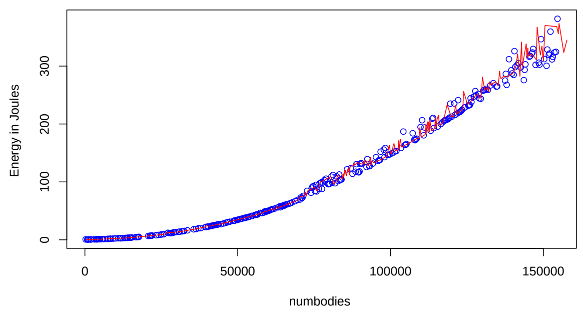


(a) SVM.

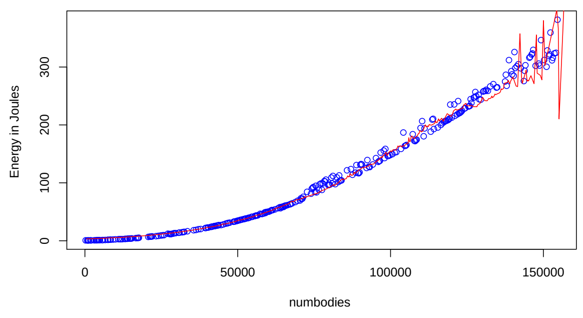


(b) DynSVM.

Figure 7: Predicted vs. Observed E_{Pack} in Joules for CUDA Nbody (v.7.5) Simulation - DynSVM and SVM.

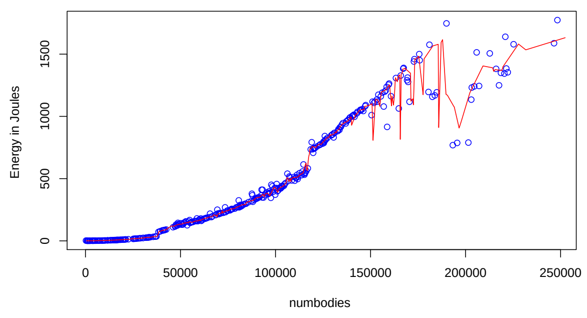


(a) DynLRM.

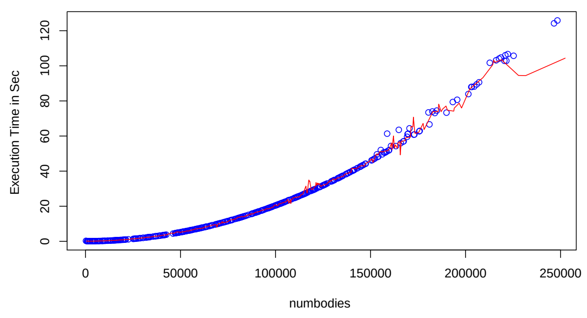


(b) LRM.

Figure 8: Predicted vs. Observed E_{Pack} in Joules for CUDA Nbody (v.7.5) Simulation - DynLRM and LRM.



(a) Energy Prediction (Joules)



(b) Execution Time Prediction (Sec.)

Figure 9: Predicted vs. Observed E_{Pack} in Joules for CUDA Nbody (v.7.5) Simulation on Sandybridge Machine - Quadro1000M.

5.3 R^2 Error - RFM, SVM, and LRM

The modeling errors obtained while predicting the energy consumption and the execution time of a section of CUDA code variants (the testing dataset) using the classical modeling mechanisms, such as, RFM, SVM, and LRM were reviewed. Table 2 lists the R^2 error values in percentage when CUDA applications were tested on the representative GPGPUs.

It could be noticed that the prediction results for E_{GPU} were poor for three models - 49.74 percentage for LRM, 67.41 (RFM), and 41.76 for SVM. This occurred due to the fact that the independent variables that are used for predicting the energy consumption of uncore elements on applications were not properly correlated with the dependent variable. In most cases, RFM prediction was better than the other prediction mechanisms.

5.4 Influence of Dynamism - DynRFM, DynSVM, DynLRM

To validate the importance of including dynamism in models, dynamic models are experimented using Particle simulation on Ivybridge processor machine. Dyanamic models remain successful as training dataset are mostly time-series in CUDA code variants.

From Table 3, we could infer that the dynamic models surpasses from the classical models with significant performance prediction improvements, especially in DynLRM. For instances, DynLRM had an energy prediction improvement of 50.26 percentages and the execution time prediction had the prediction improvement of 61.23 percentages. DynRFM also showed 15.79 percentage of prediction improvement for E_{GPU}. However, the execution time prediction showed very less improvement for DynRFM and DynSVM modeling approaches.

Table 2: R^2 Values for CUDA Applications - Comparision of LRM, RFM, and SVM

Prediction (Application)	E_{Pack}(J) GT630-R^2 in %	E_{Pack}(J) Q1000M-R^2 in %	E_{GPU}(J) GT630-R^2 in %
Energy Prediction for Nbody Simulation			
LRM	97.3	96.37	95.58
RFM	99.5	98.95	98.87
SVM	99.15	98.71	99.29
Energy Prediction for Particle Simulation			
LRM	99.89	99.99	49.74
RFM	99.72	97.99	67.41
SVM	98.71	80.66	41.76
ExecTime Prediction for Nbody Simulation			
LRM	96.68	90.09	94.87
RFM	99.83	99.03	98.72
SVM	99.35	98.98	99.13
ExecTime Prediction for Particle Simulation			
LRM	98.89	99.98	38.77
RFM	99.91	98.03	99.86
SVM	98.41	80.66	96.21

Table 3: R^2 Values for CUDA Applications - Influence of Dynamism to LRM, RFM, and SVM

Prediction (Approach)	Classical Approach (RFM,...)	Dynamic Approach (DynRFM...)	Improvement in %
Energy Prediction			
Linear	49.74	100	50.26
Random Forests	67.41	80.05	15.79
Support Vector	41.76	49.77	16.09
ExecTime Prediction			
Linear	38.77	100	61.23
Random Forests	99.86	99.9	0.04
Support Vector	96.21	96.49	0.29

5.5 Model Specific Variations

In addition to the prediction improvements happened due to dynamic models, it was a pivotal decision to delve into the prediction results while varying the model specific parameters. In general, RFM could be efficiently predicted by varying the number of RFTrees, number of independent variables for deciding the split of branches, and so forth; similarly, SVM has its own parameters, such as, kernels, epsilon criteria, and so forth, for improving the performance prediction mechanisms.

5.5.1 DynRFM Variants

Figure 10 and Figure 11 illustrate the need for varying the RFM model specific parameters, namely, RFTrees and number of independent variables. On investigating Particle simulation, the energy prediction of CUDA code variants gradually improved while increasing the number of independent variables listed in paragraph 5.1.2. In addition, it was observed that the prediction improvement due to DynRFM was gradual when the number of RFTrees for Particle simulation was increased (refer the R^2 values for RFTrees=1000 and RFTrees=10000 in Figure 10 and Figure 11).

5.5.2 DynSVM Variants

Additionally, we studied the prediction improvements that hap-

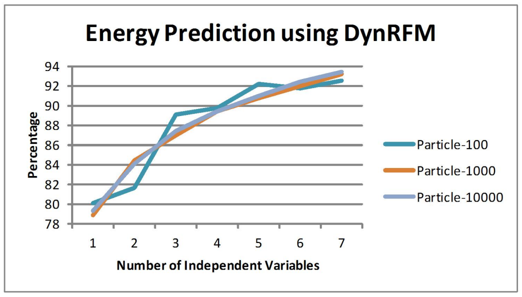


Figure 10: R^2 Energy Prediction Improvement using RFM Variants

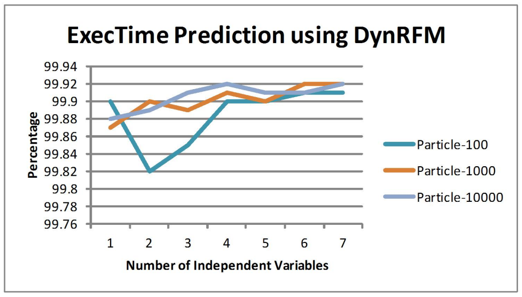


Figure 11: Execution Time Prediction Improvement using RFM Variants

pened in DynSVM modeling approach while varying the kernel functions of DynSVM. As mentioned earlier, SVM had four basic kernels, such as, linear, polynomial, sigmoid, and radial, which influence the prediction results of testing data.

On substituting different kernels on the process of modeling the candidate CUDA applications, it was learned that linear kernel would fit very well for our prediction problem. Sigmoid and Polynomial kernels showed scanty results. For instance, the R^2 values for the execution time modeling using the polynomial kernel of DynSVM was only 43.03 percentage. Table 4 lists the R^2 values obtained for the CUDA applications.

Table 4: R^2 Values for CUDA Applications on Introducing DynSVM Variants

Simulation	Prediction	DynSVM (Kernels)			
		Linear	Polynomial	Sigmoid	Radial
Nbody	Energy	99.64	57.73	*	99.31
Nbody	ExecTime	99.77	70.43	*	99.17
Partilce	Energy	99.68	65.33	*	49.77
Particle	ExecTime	98.50	43.03	*	96.49

6. CONCLUSION

Energy consumption issue is still remaining as a challenge for HPC application developers. On reaching the goals of exa-scale computing, CUDA-based HPC application developers thrive hard to attain the full computing potential of emerging parallel machines.

One among the available research efforts is to autotune CUDA applications using tools so that the users could utilize the best optimal combinations for the applications. However, tuning all available combinations is a challenging task. On the contrary, prediction mechanisms could be applied for autotuning.

In this paper, we had endeavored several prediction mechanisms, such as, LRM, SVM, RFM, DynLRM, DynSVM, and DynRFM for CUDA applications. In addition, a few prediction algorithm specific parameter variations were also studied. The proposed energy/performance prediction framework for CUDA applications was tested with Nbody and Particle simulations. Our experimental results have shown a performance prediction improvement of over 50.26 to 61.23 percentages.

7. ACKNOWLEDGMENTS

This work is funded by the Department of Science and Technology of India under the Indo-Austrian PPP Scheme - Engineering Sciences division. The authors thank DST (India) and FWF (Austria) for funding this research work.

8. REFERENCES

[1] Alcides Fonseca, Bruno Cabral, ÃĘminiumGPU: An Intelligent Framework for GPU Programming, Facing the Multicore-Challenge III, Volume 7686 of the series Lecture Notes in Computer Science, pp 96-107, 2013.

[2] Andrew Kerr, Gregory Diamos, and Sudhakar Yalamanchili. Modeling GPU-CPU workloads and systems. In Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU-3). ACM, New York, NY, USA, 31-42. DOI=10.1145/1735688.1735696 http://doi.acm.org/10.1145/1735688.1735696.

[3] Ariel A., Fung W.W.L., Turner A.E., Aamodt T.M., "Visualizing complex dynamics in many-core accelerator architectures," 2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS), pp.164-174, 28-30 March 2010, doi: 10.1109/ISPASS.2010.5452029.

[4] Ashwin M. Aji, Mayank Daga, Wu-chun Feng, CampProf: A Visual Performance Analysis Tool for Memory Bound GPU Kernels, in https://vtechworks.lib.vt.edu/bitstream/handle/10919/19729/CampProf-TechReport.pdf?sequence=3&isAllowed=y, accessed in Dec. 2015.

[5] D. A. Bacigalupo, S. A. Jarvis, L. He, D. P. Spooner, D. N. Dillenberger and G. R. Nudd, "An Investigation into the Application of Different Performance Prediction Methods to Distributed Enterprise Applications," Journal of Supercomputing, vol. 34, no. 2, 2005.

[6] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski and M. Schulz, "A regression- based approach to scalability prediction," in 22nd Annual International Conference on Supercomputing, 2008.

[7] Barra Simulator, https://code.google.com/p/barra-sim/, accessed on 1 Oct 2015.

[8] Boyer M., Jiayuan Meng, Kumaran K., "Improving GPU Performance Prediction with Data Transfer Modeling," 2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pp.1097-1106, 20-24 May 2013, doi: 10.1109/IPDPSW.2013.236.

[9] AutoTuning Seminar, http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=13401, accessed on Oct, 2015.

[10] J. Brehm and P. Worley, "Performance Prediction for Complex Parallel Applications," in 11th International Symposium on Parallel Processing, 1997.

[11] J. Cao, S. A. Jarvis, D. P. Spoon, J. D. Turner, D. J. Kerbyson and G. R. Nudd, "Performance Prediction Technology for Agent-Based Resource Management in Grid Environments," in 16th International Parallel and Distributed Processing Symposium, Washington, DC, USA, 2002.

[12] L. Carrington, A. Snavely and N. Wolter, "A performance prediction framework for scientific applications," Future Generation Computer Systems, vol. 22, no. 3, February 2006.

[13] Cyril Zeller, NVIDIA's vision for Exa-scale, http://www.teratec.eu/library/pdf/forum/2014/Presentations/SP04_C_Zeller_NVIDIA_Forum_Teratec_2014.pdf.

[14] Usman Dastgeer, Johan Enmyren, and Christoph W. Kessler, Auto-tuning SkePU: a multi-backend skeleton programming framework for multi-GPU systems. In Proceedings of the 4th International Workshop on Multicore Software Engineering (IWMSE '11), ACM, New York, NY, USA, pp. 25-32. DOI=http://dx.doi.org/10.1145/1984693.1984697, 2011.

[15] Elizabeth T, Nathan C., David A.P., John B., Barry I.P., and Dave H., Parallel cuda implementation of conflict detection for application to airspace deconfliction, J.of Supercomputing, Vol.71, No.10, pp.3787-3810, 2015, DoI:10.1007/s11227-015-1467-z.

[16] Fauzia N., Pouchet L.-N., Sadayappan, P., "Characterizing and enhancing global memory data coalescing on GPUs," 2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pp.12-22, 7-11 Feb. 2015, doi: 10.1109/CGO.2015.7054183.

[17] Haifeng Wang and Yunpeng Cao, Predicting power consumption of GPUs with fuzzy wavelet neural networks, in Parallel Computing, Vol.44, pp.18-36, doi:10.1016/j.parco.2015.02.002.

[18] Hartwig A., Blake H., Jakub K., Piotr L., and Dongarra J., Experiences in autotuning matrix multiplication for energy minimization on GPUs, Concurrency and Computation: Practice and Experience, DOI: 10.1002/cpe.3516.

[19] M. A. Iverson, F. Ozguner and L. Potter, "Statistical Prediction of Task Execution Times through Analytic Benchmarking for Scheduling in a Heterogeneous Environment," IEEE Transactions on Computers, vol. 48, no. 12, pp. 1374-1379, December 1999.

[20] Jaewoong Sim, Aniruddha Dasgupta, Hyesoon Kim, and Richard Vuduc. A performance analysis framework for identifying potential benefits in GPGPU applications. In Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP '12), ACM, New York, NY, USA, 11-22. DOI=10.1145/2145816.2145819 http://doi.acm.org/10.1145/2145816.2145819.

[21] Jiayuan M. and Kevin S., A Performance Study for Iterative Stencil Loops on GPUs with Ghost Zone Optimizations, in International Journal of Parallel Programming, Vol.39, No.1, pp-115-142, DOI:10.1007/s10766-010-0142-5.

[22] Junjie Lai and Andre Seznec. Performance upper bound analysis and optimization of SGEMM on Fermi and Kepler GPUs. In Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO) (CGO '13), IEEE Computer Society, Washington, DC, USA, 1-10. DOI=10.1109/CGO.2013.6494986 http://dx.doi.org/10.1109/CGO.2013.6494986.

[23] Kamil S., Chan C., Oliker L., Shalf J., and Williams S., "An auto-tuning framework for parallel multicore stencil computations," 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), pp.1-12, 19-23 April 2010, doi: 10.1109/IPDPS.2010.5470421.

[24] N. Kapadia, J. Fortes and C. Brodley, "Predictive application-performance modeling in a computational grid environment," in 8th International Symposium on High Performance Distributed Computing, 1999.

[25] H. Li, D. Groep, J. Templon and L. Wolters, "Predicting Job Start Times on Clusters," in International Symposium on Cluster Computing and the Grid, 2004.

[26] Monakov A, Lokhmotov A, Avetisyan A, Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures, LNCS, Springer, pp.111-125, http://dx.doi.org/10.1007/978-3-642-11515-8_10, 2010.

[27] F. Nadeem, Y. Murtaza, R. Prodan and T. Fahringer, "International Conference on e-Science and Grid Computing," in Soft Benchmarks-based Application Performance Prediction using a Minimum Training Set, Amsterdam, Netherlands, 2006.

[28] NVIDIA GPGPU applications, http://www.nvidia.com/content/gpu-applications/PDF/GPU-apps-catalog-mar2015.pdf, accessed on 30 Sep. 2015.

[29] 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.

[30] Shajulin Benedict, Rejitha R.S., Phillip G., Radu Prodan, Thomas Fahringer, 'Energy Prediction of OpenMP Applications using Random Forest Modeling Approach', in iWAPT2015 @ IPDPS 2015, DOI:10.1109/IPDPSW.2015.12, pp.1251-1260, 2015.

[31] Shajulin Benedict, Rejitha R.S., Preethi B., Bright C., and Judyfer W.S., 'Energy Analysis of Code Regions of HPC Applications using EnergyAnalyzer Tool', (in press) Int. Journal of Computational Science and Engineering, InderScience publishers, 2015.

[32] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, and Kevin Skadron. A performance study of general-purpose applications on graphics processors using CUDA. J. Parallel Distrib. Comput. Vol.68, No. 10 (October 2008), pp. 1370-1380. DOI=10.1016/j.jpdc.2008.05.014 http://dx.doi.org/10.1016/j.jpdc.2008.05.014.

[33] Shuaiwen Song, Chunyi Su, Rountree B., Cameron K.W., "A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures," 2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS), pp.673-686, 20-24 May 2013, doi: 10.1109/IPDPS.2013.73.

[34] Sierra-Canto X., Madera-Ramirez, F., Uc-Cetina, V., "Parallel Training of a Back-Propagation Neural Network Using CUDA," 2010 Ninth International Conference on in Machine Learning and Applications (ICMLA), pp.307-312, 12-14 Dec. 2010 doi: 10.1109/ICMLA.2010.52.

[35] W. Smith, I. Foster and V. Taylor, "Predicting application run times with historical information," Journal of Parallel and Distributed Computing, vol. 64, no. 9, pp. 1007-1016, September 2004.

[36] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia and A. Purkayastha, "A framework for performance modeling and prediction," in Supercomputing Conference, 2002.

[37] S. Sodhi, J. Subhlok and Q. Xu, "Performance prediction with skeletons," Cluster Computing, vol. 11, no. 2, pp. 151-165, 2008.

[38] R. Susukita, H. Ando, M. Aoyagi, H. Honda, Y. Inadomi, K. Inoue, S. Ishizuki, Y. Kimura, H. Komatsu, M. Kurokawa, K. J. Murakami, H. Shibamura, S. Yamamura and Y. Yu, "Performance prediction of large-scale parallell system and application using macro-level simulation," in Supercomputing Conference, 2008.

[39] A. Takefusa, O. Tatebe, S. Matsuoka and Y. Morita, "Performance analysis of scheduling and replication algorithms on Grid Datafarm architecture for high-energy physics applications," in 12th International Symposium on High Performance Distributed Computing, 2003.

[40] V. Taylor, X. Wu, J. Geisler and R. Stevens, "Using Kernel Couplings to Predict Parallel Application Performance," in 11th IEEE International Symposium on High Performance Distributed Computing, Washington, DC, USA, 2002.

[41] X. Wu, V. Taylor and J. Paris, "A Web-based Prophesy Automated Performance Modeling System," in International Conference on Web Technologies, Applications and Services, 2006.

[42] Tingxing Dong, Dobrev, V., Kolev, T., Rieben, R., Tomov, S., Dongarra, J., "A Step towards Energy Efficient Computing: Redesigning a Hydrodynamic Application on CPU-GPU," in Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, pp.972-981, 19-23 May 2014 doi: 10.1109/IPDPS.2014.103.

[43] A. Tirado-Ramos, G. Tsouloupas, M. Dikaiakos and P. Sloot, "Grid Resource Selection by Application Benchmarking for Computational Haemodynamics Applications," in International Conference on Computational Science, 2005.

[44] Vedran M., Martina H.D., and NataÅąa H.B, Optimizing ELARS Algorithms using NVIDIA CUDA Heterogeneous Parallel Programming Platform, ICT Innovations 2014, pp.135-144, doi:10.1007/978-3-319-09879-1_14.

[45] F. Vraalsen, R. A. Aydt, C. L. Mendes and D. A. Reed, "Performance Contracts: Predicting and Monitoring Grid Application Behavior," in 2nd International Workshop on Grid Computing, 2001.

[46] Y. Wu, L. Liu, J. Mao, G. Yang and W. Zheng, "An analytical model for performance evaluation in a computational grid," in 3rd workshop on High performance computing in China, 2007.

[47] Xiangzheng Sun, Yunquan Zhang, Ting Wang, Xianyi Zhang, Liang Yuan, Li Rao, "Optimizing SpMV for Diagonal Sparse Matrices on GPU," 2011 International Conference on in Parallel Processing (ICPP), pp.492-501, 13-16 Sept. 2011, doi: 10.1109/ICPP.2011.53.

[48] L. T. Yang, X. Ma and F. Mueller, "Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution," in Supercomputing, 2005.

[49] E. J. H. Yero and M. A. A. Henriques, "Contention-sensitive static performance prediction for parallel distributed applications," Performance Evaluation, vol. 63, no. 4, pp. 265-277, May 2006.

[50] Yooseong Kim and Shrivastava A., CuMAPz: A tool to analyze memory access patterns in CUDA, in Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE, pp. 128 - 133, 2011.

[51] Yuji Kubota and Daisuke Takahashi, Optimization of Sparse Matrix-Vector Multiplication by Auto Selecting Storage Schemes on GPU, LNCS, in ICCSA 2011, pp.547-561, 2011.