Available on-line at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Investigating and Exploiting Application Dynamism For Energy-Efficient Exascale Computing

Venkatesh Kannan[a,*], Lubomír Říha[b], Michael Gerndt[c], Anamika Chowdhury[c], Ondřej Vysocký[b], Martin Beseda[b], Horák David[b], Radim Sojka[b], Jakub Kruzik[b], Michael Lysaght[a]

[a]Irish Centre for High-End Computing, Dublin, Ireland
[b]IT4Innovations, Ostrava, Czech Republic
[c]Institute für Informatik, Technical University of Munich, Germany

Abstract

READEX is a EU Horizon 2020 FET-HPC project whose objective is to exploit the dynamism found in high-performance computing applications at runtime to achieve efficient computation on Exascale systems. In this paper, we describe the use of the READEX methodology to investigate dynamic behaviour of PRACE-relevant applications at runtime and describe how such application dynamism can be exploited to tune a range of application-, system software- and hardware-level parameters for improved performance and energy efficiency on current and future European extreme-scale systems.

1. Introduction

High performance computing (HPC) is a major driving force for European research and innovation in many scientific and industrial domains. The applications in these areas are highly complex, and demand high performance and efficient execution. As a result of this growing need for computational performance, the energy consumption of the HPC systems has continued to increase. This is a cause for concern because the energy requirements of near-future European extreme-scale systems will be a major factor of the total cost of owning the HPC system. Therefore, it is crutial to improve the energy-efficiency of the applications that run on these systems.

A significant source of improvement for applications is that they commonly exhibit dynamic resource requirements. This may stem from different regions in the application that are executed or changes in the workload at runtime. Consequently, such dynamism in an application presents opportunity to tailor the utilisation of resources in the HPC system based on the requirements of the application at runtime. The READEX (Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing) project leverages this approach to deliver improved performance and energy-efficiency when executing applications on current and future extreme-scale systems. The goal of the READEX project is to create a tool suite that

1. Identifies existing dynamism in an application to determine its tuning potential.
2. Determines the configurations for different hardware-, system software- and application-level tuning parameters that suit different scenarios that may arise during the application's execution.
3. Applies the best configuration for the tuning parameters during the application's runtime.

In this paper, we describe how, as a part of PRACE WP7, we have chosen to investigate an alpha prototype of the READEX tool suite to identify existing dynamism in an application to determine its tuning potential. When analysing a given application, the READEX tool suite quantifies the application dynamism using *dynamism metrics* such as compute intensity and execution time of the application. The objective of the tool suite is to switch the values for different tuning parameters at runtime to optimal configurations based on the observed application dynamism. We also present the preliminary results from our tuning experiments to demostrate the potential savings that may be achieved using the READEX approach.

In Section 2., we present an overview of the READEX tool suite and the workflow to describe its use. In Section 3., we use preliminary results to demonstrate how the technologies developed in the READEX project enable quantifying the dynamism in an application and the potential gain demonstrated by our tuning experiments. In Section 4., we describe the reporting structure that has been created to summarise and present the results of the application dynamism analysis experiments that are performed in READEX. A wider set of tuning parameters that are targeted by READEX and related work are discussed in Section 5.

* Corresponding author: e-mail. venkatesh.kannan@ichec.ie tel. +353-1-524-1608 fax. +353-1-764-5845

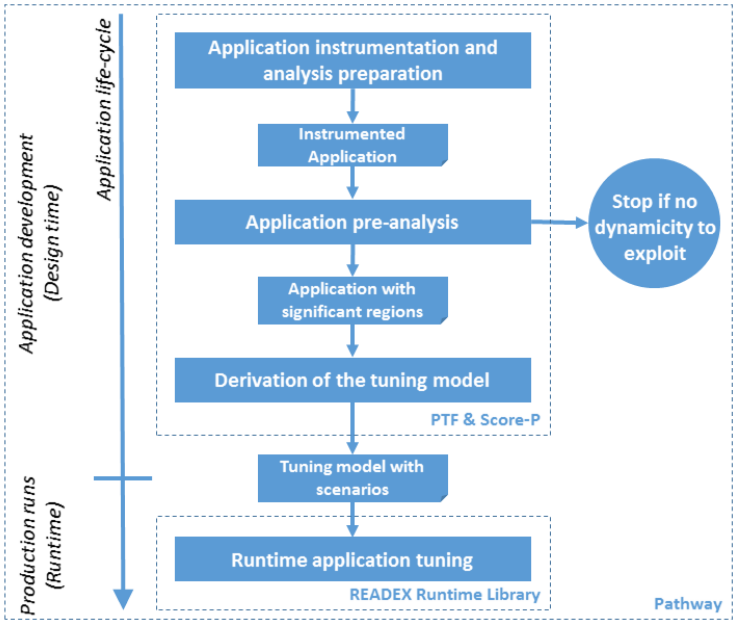

Fig. 1. Overview of READEX workflow

2. Overview of READEX Methodology

2.1. Dynamism Metrics

The application dynamism metrics that are presently measured and used in the READEX tool suite are execution time and compute intensity. While the semantics of execution time is straightforward, compute intensity is a metric that is used to model the behaviour of an application based on the workload imposed by it on the CPU and the memory. In the alpha prototype of the tool suite, compute intensity is calculated using the following formula and is analogous to the operational intensity used in the roofline model [13]:

$$Compute\ intensity \quad = \quad \frac{Total\ number\ of\ instructions\ executed}{Total\ number\ of\ L3\ (LLC)\ cache\ misses}$$

Similar to operational intensity, compute intensity can directly dictate the tuning of two hardware parameters associated with the processor: core frequency and uncore frequency. For instance, a low compute intensity may indicate an application that is more memory intensive, which results in increased L3 (LLC) cache misses. Since this would cause increased traffic between the L3 cache and the main memory, it will be desirable to increase the uncore frequency. On the other hand, a high compute intensity may indicate an application that is more computation intensive. In this case, it will be desirable to increase the frequency of the CPU cores.

2.2. Objective Values

By measuring the variations in these dynamism metrics in an application, the READEX tool suite aims to optimise execution time or energy consumption (or a combination of both) of the application. These are considered to be the *objective values* that are to be optimised in the READEX project.

2.3. Tuning Parameters

The objective values are optimised by switching the configuration of the *tuning parameters* based on the observed values of the dynamism metrics. For instance, the tuning parameters whose configurations can be explored using tuning experiments performed in the alpha prototype of the READEX tool suite are processor core frequency, uncore frequency of the processor socket and the number of OpenMP threads used to run the application.

2.4. READEX Workflow

An overview of the READEX workflow in which the READEX tool suite is applied on an application in two phases is illustrated in Fig. 1 and explained below:

1. **Design-time Analysis (DTA)**
   This is performed during application development and is composed of the following steps:

(a) Prepare the application for analysis by instrumenting regions of code that contribute significantly to objective values – execution time and energy consumption – that are to be optimised. This instrumentation is done by adding annotations, both automatically and by letting the user add application-domain knowledge to expose parameters that define the dynamic behaviour of the application using a new programming paradigm being developed in the project.

(b) This instrumented application is analysed to first identify existing dynamism that can be exploited by tuning hardware, system-software and application parameters. If a significant amount of dynamism is detected, then the regions of code that express this dynamism, referred to as *significant regions*, are identified and enlisted.

(c) Following this, experiments are performed to apply various configurations to the tuning parameters in order to estimate the objective values (execution time and energy consumption) for each runtime instance of each significant region, referred to as a *runtime situation* (*rts*).

(d) The results of these experiments are used to cluster *rts*s with similar objective values into a *scenario*.

(e) The results of the design-time analysis are aggregated into a *tuning model* which is composed of the specifications for

- *Scenarios*: Groups of *rts*s which are found to require the same best configuration for the tuning parameters.
- *Configurations*: Values for the tuning parameters, with one configuration for each scenario listed in the tuning model.
- *Classifier*: A methodology to map a given *rts* into one of the scenarios listed in the tuning model.
- *Selector*: A methodology to identify the configuration best suited for a given scenario.

2. **Runtime Application Tuning (RAT)**
This is performed during the production run of an application and is composed of the following steps:

(a) During the production run of the application, each encountered *rts* is mapped into a scenario using the classifier.

(b) Following this, the best configuration of the tuning parameters for the current scenario is identified using the selector, and the tuning parameters are switched to the pre-computed values listed in the tuning model.

(c) Upon encountering an *rts* that was not seen and classified at DTA, the configuration for a default scenario that is guaranteed to satisfy any *rts* is used.

During the course of the READEX project, RAT will be extended to include a calibration mechanism for the tuning model in which the classification, selection and configuration switching mechanisms may be updated based on analysis performed by the READEX tool suite during the production run.

More details on the workflow of the READEX tool suite are available in [8].

In the READEX tool suite, DTA is performed by the Periscope Tuning Framework (PTF) in conjunction with Score-P which provides the instrumentation and measurement infrastructure. RAT is performed by a newly developed READEX Runtime Library (RRL) in conjunction with Score-P. The RRL is also used during DTA to perform the tuning parameter switching when searching for the best configurations using PTF.

3. Investigating Application Dynamism

In this section, we demostrate how technologies developed in the READEX project allow the analysis of an application during the "application pre-analysis" step in the READEX workflow. The objective is to identify different types of dynamism that may exist in the application, particularly *inter-* and *intra-phase dynamism*.

To explain this, consider the snippet of C code shown in Fig. 2. Here, the code regions of interest are the body of the `for`-loop and the calls to functions `laplace()` and `fftw()` due to their expected higher resource requirements in comparison with the rest of the code. Being the body of a main iterative computation in the application, the `for`-loop body is referred to as a *phase region*. In this phase region, the functions `laplace()` and `fftw()` are the *significant regions*. In this context, the READEX tool suite targets identification of two types of dynamic behaviour:

- *Inter-phase dynamism*: This is exhibited if the dynamism metrics vary significantly for different runtime instances (phases) of a phase region. For example, inter-phase dynamism is present if the execution time measured for different iterations of the `for`-loop in Fig. 2 is significantly different.
- *Intra-phase dynamism*: This is exhibited if the dynamism metrics vary significantly for different runtime instances of the significant regions in a phase. For example, intra-phase dynamism is present if the execution time or compute intensity of `laplace()` and `fftw()` vary significantly in an iteration of the `for`-loop.

In Section 3.1., we use a mini-application, miniMD from the Mantevo benchmark suite [6], to illustrate the presence of dynamism using execution time and compute intensity as the dynamism metrics. The relevance of miniMD stems from it being a light-weight version of NAMD which is a widely-used molecular dynamics application in the European HPC community. In Section 3.2., we briefly present the technologies that are being developed in the READEX tool suite to instrument an application and the dynamism identified by the tool suite using miniMD. This demostrates and evaluates the techniques developed in the READEX project to identify

```
int main(void)
{
  // Initialise application
  // Initialise experiment variables

  int num_iterations = 100;

  for (int i = 1; i <= num_iterations; i++)
  {
    // Start of phase region

    laplace();              // Significant region
    residue = reduction();  // Insignificant region
    fftw();                 // Significant region

    // End of phase region
  }

  // Post-processing:
  // Write noise matrices to disk for visualisation
  // Teriminate application

  return 0;
}
```

Fig. 2. Code sample with phase and significant regions

existing dynamism in an application for improving its performance and energy consumption. In Section 3.3., we present the preliminary results of the savings achieved from applying tuning actions in response to the identified dynamism for the total FETI solvers (PERMON [7] AND ESPRESO [9]), BLAS routines and the AMG2013 Proxy application [1]. It is key to note that FETI solvers, such as PERMON and ESPRESO, are also of big interest and relevance to the European HPC community. These reported savings serve to support the goals of the READEX project and the tool suite which is under development. The results of comparing the savings observed from the tuning experiments will be compared in future to those achieved by runtime tuing performed automatically by the READEX tool suite.

3.1. Manual Analysis of Dynamism

We use the simple miniMD application to illustrate dynamism that may exist in applications. Firstly, we identify the phase region that is of interest in the application. Fig. 3 presents the `run(...)` function that contains the main iterative computation in miniMD. Here, the body of the `for`-loop is the phase region which performs the simulation for the number of times (`ntimes`) specified as a part of the input data provided to miniMD. The phase region contains calls to multiple functions among which some are identified as *significant regions* since their execution times are a significant part of the total application run time. The remaining regions are considered to be *insignificant regions*.

As a part of the manual dynamism analysis, our next objective is to identify variations in the dynamism metrics of interest to us in this phase region. Presently, we use variations in execution time and computational itensity to quantify the existing dynamism. For compute intensity, we use the PAPI (Performance Application Programming Interface [3]) counters `PAPI_INS_TOT` and `PAPI_L3_TCM` to calculate the compute intensity as explained in Section 1. for each iteration of the `for`-loop. An example of the variation in execution time and compute intensity measured in miniMD is illustrated in Fig. 4. This experiment was run with 8 MPI processes on 8 cores of an Intel Xeon 2680v3 processor with 2.5 GB memory. Consequently, we observe the following:

- The execution time and compute intensity spikes once every `k` iterations of the `for`-loop. Here, `k = 10`.
- The observed spikes are due to the execution of the body of the `if ((n+1) % neighbour.every)` block in the `run(...)` function.

Here, the frequency of the spike in execution time and compute intensity is decided by `neighbour.every` which is an input data provided to miniMD.

Thus, this illustrates the existence of inter-phase dynamism in the miniMD application. Following this, our objective is to demonstrate how the tool suite developed in the READEX project supports identification of dynamism in applications. This is presented using the miniMD example in the following section.

3.2. Analysis of Dynamism using READEX Technologies

One of the components developed in the READEX tool suite, `readex-dyn-detect`, is targeted to measure and quantify the variation in the dynamism metrics of interest in a given application. For this, the first step is to identify and annotate the phase region in the application using Score-P. An example of the annotated phase region of the miniMD application is presented in Fig. 5.

```
void Integrate::run(...)
{
  // Initialise

  for (n = 0; n < ntimes; n++)
  {
    // Start of phase region
    ...
    initialIntegrate();        // Insignificant region
    ...
    if ((n+1) % neighbour.every)
    {
            // Reneighbour atoms
    }
    else
    {
            // Computation for atoms
    }
    ...
    force->compute(...);        // Significant region
    ...
    // End of phase region
  }
}
```

Fig. 3. Code snippet of miniMD phase region

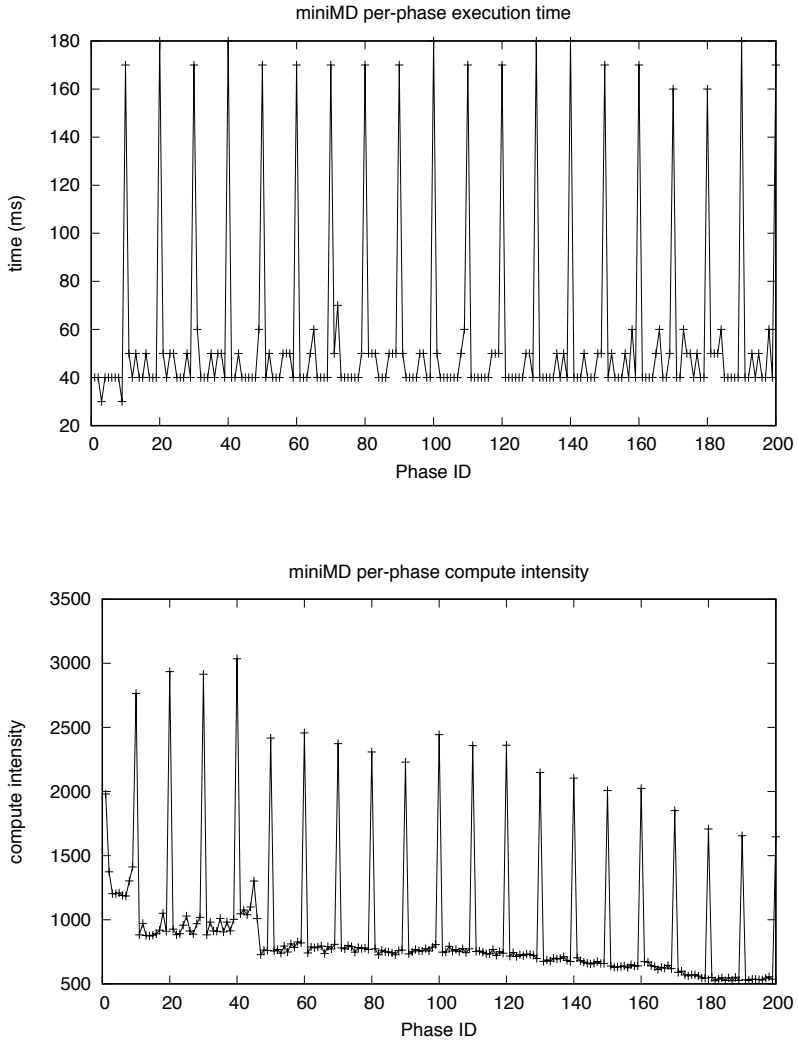

Fig. 4. Inter-phase dynamism observed in miniMD using manual analysis

```
void Integrate::run(...)
{
  // Initialise

  SCOREP_USER_REGION_DEFINE(R1)
  for (n = 0; n < ntimes; n++)
  {
    // Start of phase region
    SCOREP_USER_OA_PHASE_BEGIN(R1, "INTEGRATE_RUN_LOOP", 2)
    ...
    initialIntegrate(); // Insignificant region
    ...
    if ((n+1) % neighbour.every)
    {
      // Reneighbour atoms
    }
    else
    {
      // Computation for atoms
    }
    ...
    force->compute(...);  // Significant region
    ...
    // End of phase region
    SCOREP_USER_OA_PHASE_END(R1)
  }
}
```

Fig. 5. Code snippet of miniMD phase region annotated with Score-P

`readex-dyn-detect` identifies and quantifies the inter- and intra-phase dynamism using execution time and compute intensity as dynamism metrics. Variation in the execution time across *rts*s of the phase region indicates inter-phase dynamism. Variation in the execution time of significant regions across *rts*s indicates intra-phase dynamism. Furthermore, different compute intensity, i.e., compute vs. memory bound, of different significant regions also indicates intra-phase dynamism. To quantify the dynamism that is identified, statistical information about the performance metrics is measured via Score-P that includes the number of samples, minimum, maximum, average, and deviation for each metric.

3.2.1. Intra-phase dynamism

Intra-phase dynamism detection by `readex-dyn-detect` is based on measuring the execution time and compute intensity. Therefore, the total number of instructions and L3 (LLC) cache misses are measured via PAPI.

The tool then analyzes for each significant region the variation in the execution time characteristics. It computes the standard deviation relative to the mean execution time of the region in percent ($deviation_r$) and relative to the mean execution time of the phase ($deviation_p$) as described in Equations 1 and 2 below, respectively. The values characterize how significant the variation in the execution time is for the region and phase execution respectively.

$$deviation_r^{reg} = \frac{dev\_t_{incl}^{reg}}{mean\_t_{incl}^{reg}} * 100 \tag{1}$$

$$deviation_p^{reg} = \frac{dev\_t_{incl}^{reg}}{mean\_t_{incl}^{phase}} * 100 \tag{2}$$

The variation is considered significant if it is larger than a threshold v_t. To decide whether this leads to significant dynamism, the tool computes the computational weight of the region, i.e., its percentage on the phase execution time, according to the formula

$$weight = \frac{t_{incl}^{reg}}{t_{incl}^{phase}} * 100. \tag{3}$$

If the region's execution time variation is significant and its weight is larger than a threshold v_w then the tool reports the presence of intra-phase dynamism due to that significant region.

Another source of intra-phase dynamism is the variation of other characteristics across different significant regions. The tool currently supports a comparison based on the compute intensity of significant regions. It is based on the number of total retired instructions and the number of L3 cache misses. The former is used as a measure for the work done and the latter for the amount of data transferred between memory and the L3 cache. It should be noted that this measure of transferred data is coarse, since it does not take into account all writes to memory as well as, for example, hardware prefetching. The tool checks all significant regions with a

weight above v_w. It compares the compute intensity of those regions and reports intra-phase dynamism if the deviation is larger than a threshold v_i.

3.2.2. Inter-phase dynamism

Finally, `readex-dyn-detect` investigates whether there is significant inter-phase dynamism in the application. This analysis is based on the execution time of the phase region using the statistical performance metrics obtained from Score-P. The tool computes the relative deviation of the execution times of the phase region $deviation_p^{phase}$. If this deviation is larger than the threshold v_t, then the tool reports inter-phase dynamism.

3.2.3. Result for miniMD

An example summarising the analysis result of `readex-dyn-detect` for the miniMD application is shown in Fig. 6. The results include the list of significant regions that are identified along with statistics that include the minimum, maximum and standard deviation of the dynamism metrics for the significant regions and phases from the dynamism analysis experiments. Also included are the weight of the dynamism metric for each significant region as a percentage of the metric measured for the phase as described above. Based on this, the types of dynamism identified in the application are summarised.

For the miniMD application, the dynamism analysis by READEX identifies the available inter-phase dynamism illustrated in Fig. 4. It also identifies intra-phase dynamism for routine `compute_halfneigh` due to variation in the execution time and for functions `borders`, `compute_halfneigh`, and `binatoms` due to variation in their compute intensity characteristics.

3.3. Savings from Tuning Experiments

In order to evaluate the savings achieved on the objective values, the READEX tool suite has developed a tool called `MERIC` [2] which measures resource consumption of significant regions inside the evaluated applications. The savings are measured in two parts, static and dynamic savings, using the measurements obtained with default settings (2.5 GHz processor core frequency, 3.0 GHz uncore frequency and 1 OpenMP thread) for the tuning parameters as the baseline. The static savings are measured by setting different values for the tuning parameters for each tuning experiment. In each experiment, the dynamic savings are measured by setting optimal configurations for the tuning parameters as the dynamism varies for different significant regions.

`MERIC` wraps a list of libraries, that can read and update environment settings. The list of currently supported libraries and runtime systems include the `cpufreq` on Linux, the `x86_adapt` on Taurus [5], `OpenMP` and `Cilk++`. These are used to tune parameters such as processor core frequency, uncore frequency, number of OpenMP threads and number of active cores, respectively.

To monitor the hardware CPU counters and power consumption, `MERIC` uses `HDEEM` [5], Linux `perf_event` and `PAPI` [3]. The `HDEEM` energy measurement system is developed by TU Dresden and the Atos (previously Bull) company. It can provide energy measurements of different granularity: the node level, the CPU socket level and DRAM level. It is considered to be more accurate than Intel RAPL [4] counters that approximate the energy consumption on Intel processors.

Currently, `MERIC` the users to manually select the regions of interest in the code. As the READEX tool suite is developed, the `MERIC` tool will automatically obtain the list of significant regions from the results of `readex-dyn-detect` to be used in the tuning experiments.

3.3.1. Energy Savings Evaluation for Selected Applications

In this paper, we present and discuss the energy savings achieved for different benchmark applications by applying the READEX methodology. This is performed using the `MERIC` tool which can currently measure savings either for energy consumption or execution time. In the next versions, the `MERIC` and READEX tools will be improved to use a combination of both execution time and energy consumption as a single objective function to be optimised.

The results of the overall energy savings measured using `MERIC` on a set of applications is presented in Table 1. Here, PERMON [7] and ESPRESO [10] are iterative total FETI solvers. For PERMON the the number of processor cores used is statically tuned through the number of MPI processes per node and the processor core frequency is dynamically tuned. For ESPRESO the processor core frequency and number of OpenMP threads are both dynamically tuned. All these tests were performed on a single socket of a dual socket compute node. Consequently, this increases the baseline consumption of energy consumption. Thus, the dynamism can be potentially higher if both sockets are used. In future, these tests will be performed to obtain the measurements on both sockets.

In the case of the Algebraic Multigrid (AMG2013) solver ProxyApp we were able to achieve static/dynamic savings 11.42/1.43% for AMG2013 by setting the optimal core frequency and statically tuning the number of MPI processes.

The energy savings evaluation of selected sparse and dense BLAS routines performed on one computational node was obtained by dynamically tuning the processor core frequency. The results are summarised in Table 2. In this case a significant energy savings up to 23% is observed and the optimal core frequency varies from 1.2 GHz to 2.5 GHz. So we observe that, for instance, if an application uses a combination of dense BLAS

```
Granularity threshold: 0.010000
There is a phase region

Granularity of PHASE_REGION: 0.044406
Granularity of ForceEAM::compute: 0.019962
Granularity of ForceEAM::compute_halfneigh: 0.019951
Granularity of Comm::borders: 0.159099
Granularity of Neighbor::build: 0.341959
Granularity of Neighbor::binatoms: 0.200335

Candidate regions are:
    PhaseRegion
    ForceEAM::compute
    ForceEAM::compute_halfneigh
    Comm::borders
    Neighbor::build
    Neighbor::binatoms

Call node:   ForceEAM::compute_halfneigh    Inclusive Time 15.6506
Parent node: ForceEAM::compute              Exclusive Time 3.92518e-05
Call node:   Neighbor::binatoms             Inclusive Time 6.83612
Parent node: Neighbor::build                Exclusive Time 2.93781e-10

 Significant regions are:
    Comm::borders
    ForceEAM::compute_halfneigh
    Neighbor::binatoms

Significant region information
==============================
Region name                 Min(t)          Max(t)       Time Dev.(%Reg) Ops/L3miss      Weight(%Phase)
Comm::borders               0.159           0.167            1.6           1701               18
ForceEAM::compute_halfnei   0.016           0.033           12.3             85               43
Neighbor::binatoms          0.176           0.179            0.8            452               20

Phase information
=================
Min                 Max                 Mean                Dev.(% Phase)        Dyn.(% Phase)
0.0171156           0.526382            0.0444062           245.164              1146.84
Min                 Max                 Mean                Dev.(% Phase)        Dyn.(% Phase)
0.0171156           0.526382            0.0444062           245.164              1146.84

threshold time variation (percent of mean region time): 10.000000
threshold compute intensity deviation (#ops/L3 miss): 10.000000
threshold region importance (percent of phase exec. time): 10.000000

SUMMARY:
========
Inter-phase dynamism due to variation of the execution time of phases

Intra-phase dynamism due to time variation(%) of the following important significant regions
     ForceEAM::compute_halfneigh
Intra-phase dynamism due to variation in the compute intensity of the following important significant regions
     Comm::borders
     ForceEAM::compute_halfneigh
     Neighbor::binatoms
```

Fig. 6. Dynamism identified in miniMD using READEX tool suite

Application	Static savings [%]	Dynam. savings [%]	Total Savings [%]
PERMON TFETI	11.84	2.68	14.52
ESPRESO TFETI	7.24	5.44	12.68
BLAS ROUTINES	5-23		5-23
ProxyApps 1 - AMG2013	11.42	1.43	12.85

Table 1. Static and dynamic energy savings measured by the `MERIC` Tool

1 or 2 functions and BLAS 3 functions then one can expect significant dynamism in the application. Similar observations hold for sparse BLAS functions.

Action	Opt.freq. [GHz]	Savings [%]
AXPY	1.2	19
SpMV - CSR	1.3-1.7	11-20
SpMV - COO	1.3-1.8	8-17
DMV - FLOAT	1.2	21-23
DMV - DOUBLE	1.2	17-22
SpMM - CSR	1.8-2.4	5-8
DMM - FLOAT	1.7-1.9	2-14
DMM - DOUBLE	1.8-2.5	0-10
SpMDM - CSR-DOUBLE	1.2-1.7	12-20
SpMDM - COO-DOUBLE	1.2-1.8	13-20

Table 2. Energy savings from processor core frequency tuning for BLAS 1,2,3 routines

4. Reporting Application Dynamism

The measurements collected by the application dynamism analysis experiments performed by the READEX tool suite are logged into a READEX Applicaton Dynamism Analysis Report (RADAR). The RADAR presents the values set for the tuning parameters in the experiments and measurements of the dynamism metrics and the objective values. This is done at the three levels of granularity mentioned above: (1) total application run, (2) per phase region, and (3) per significant region as shown in Figs. 7, 8 and 9, respectively.

Application Name				
Benchmark ID				
All Phases	Default	Tuning Parameters	Core Freq (GHz)	
			Thread Count	
		Dynamism Metrics	Execution Time (s)	
			Compute intensity (Instr / L3 Miss)	
	Optimal	Tuning Parameters	Core Freq (GHz)	
			Thread Count	
		Dynamism Metrics	Execution Time (s)	
			Compute intensity (Instr / L3 Miss)	
	Savings (as % of Default)		Execution Time (s)	
			Energy (J)	

Fig. 7. RADAR – Total Application Run Perspective

In each perspective, the measured dynamism metrics are presented for the default and optimal configurations that are used for the tuning parameters.

In the phase and significant region perspectives, the minimum, maximum, average and standard deviation of the dynamism metrics are presented. Additionally, the savings achieved are presented by comparing the objective values for the optimal configuration with those for the default configuration.

In the significant region prespective, the average value of a measured dynamism metric for each significant region is also presented as a percentage of that of the phase it belongs to. Further, these values are logged in the RADAR for different settings of input data set parameters and environment parameters each of which is uniquely denoted by a "Benchmark ID".

5. Conclusions

5.1. Summary

Improved performance and energy efficiency of European HPC applications is of paramount concern for extreme-scale computing systems that are under research and development, and should be of increased concern for PRACE application enablement on the road to Exascale. One approach to address this is to optimise resource utilisation and configure the environment in accordance with the requirements of the application during its execution. The READEX project aims at developing a tool suite to exploit this approach. The technologies that are developed in READEX facilitate automatic identification of dynamic resource requirements in an

Application Name				
Benchmark ID				
Phase Region				
Default	Tuning Parameters	Core Freq (GHz)		
		Thread Count		
	Dynamism Metrics	Execution Time (s)	Min	
			Max	
			Avg	
			SD	
		Compute intensity (Instr / L3 Miss)	Min	
			Max	
			Avg	
			SD	
		Number of runs		
Optimal	Tuning Parameters	Core Freq (GHz)		
		Thread Count		
	Dynamism Metrics	Execution Time (s)	Min	
			Max	
			Avg	
			SD	
		Compute intensity (Instr / L3 Miss)	Min	
			Max	
			Avg	
			SD	
		Number of runs		
Savings (as % of Default Avg)		Execution Time (s)		
		Energy (J)		

Fig. 8. RADAR – Phase Region Perspective

Application Name				
Benchmark ID				
Significant Region				
Default	Tuning Parameters	Core Freq (GHz)		
		Thread Count		
	Dynamism Metrics	Execution Time (s)	Min	
			Max	
			Avg	
			SD	
			Avg as % Phase Default Avg	
		Compute intensity (Instr / L3 Miss)	Min	
			Max	
			Avg	
			SD	
			Avg as % Phase Default Avg	
		Number of runs		
Optimal	Tuning Parameters	Core Freq (GHz)		
		Thread Count		
	Dynamism Metrics	Execution Time (s)	Min	
			Max	
			Avg	
			SD	
			Avg as % Phase Default Avg	
		Compute intensity (Instr / L3 Miss)	Min	
			Max	
			Avg	
			SD	
			Avg as % Phase Default Avg	
		Number of runs		
Savings (as % of Default Avg)		Execution Time (s)		
		Energy (J)		

Fig. 9. RADAR – Significant Region Perspective

application and tuning hardware-, system software- and application-level parameters for optimised objective values such as performance and energy consumption.

In this paper, we have demonstrated with PRACE-relevant applications how the READEX tool suite (by way of an alpha prototype) identifies and reports on dynamism in an application. We also presented the initial results from tuning experiments that were performed on larger applications to demonstrate the potential savings that are targeted by the READEX approach. The development roadmap of the READEX tool suite is discussed in detail in a deliverable report [8].

5.2. Related Work

The ANTAREX project [12] targets efficient performance and energy consumption of applications on heterogeneous Exascale systems that are composed of multi-core CPUs and accelerators. To achieve this, a domain-specific language is designed for expressing adaptivity, energy and performance strategies to enforce runtime application tuning along with resource and power management. Consequently, an additional compilation steps is required to translate the domain-specific language into the target language.

A manual approach to dynamic tuning of parameters was described by Schöne and Molka [11]. This approach extends the VampirTrace framework with a library for users to specify optimal configurations for individual regions, which are later applied at runtime.

The READEX approach is unique and interesting due to automatic analysis and identification of dynamism in an application and its modelling, using tuning experiments performed at design time, to later perform tuning actions during the application's production run.

References

1. CORAL Benchmarks. Last accessed August 10, 2015.
2. MERIC Tool for Energy Measurement - GIT Repository. https://code.it4i.cz/xvysoc01/meric.git.
3. Performance Application Programming Interface. *http://icl.cs.utk.edu/papi/*.
4. RAPL Interface. *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3*.
5. Taurus HPC System, TU Dresden. https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/SystemTaurus.
6. The Mantevo Benchmark Suite. www.mantego.org.
7. The PERMON (Parallel, Efficient, Robust, Modular, Object-oriented, Numerical) Toolbox. http://industry.it4i.cz/en/products/permon/.
8. Michael Lysaght, Kashif Iqbal, Joseph Schuchart, Andreas Gocht, Michael Gerndt, Anamika Chowdhury, Madhura Kumaraswamy, Per Gunnar Kjeldsberg, Magnus Jahre, Mohammed Sourouri, David Horak, Lubomir Riha, Radim Sojka, Jakub Kruzik, Kai Diethelm, and Othman Bouizi. D4.1: Concepts for the READEX tool suite. Technical report, ICHEC, TUD, TUM, NTNU, IT4I, Intel, gns, 2016.
9. L. Riha, T. Brzobohaty, A. Markopoulos, O. Meca, and M. Merta. ExaScale PaRallel FETI SOlver (ESPRESO). http://espreso.it4i.cz. Last accessed January 14, 2016.
10. Lubomir Riha, Tomas Brzobohaty, Alexandros Markopoulos, Ondrej Meca, and Tomas Kozubek. Massively parallel hybrid total feti (htfeti) solver. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '16, New York, NY, USA, 2016. ACM.
11. Robert Schöne and Daniel Molka. Integrating performance analysis and energy efficiency optimizations in a unified environment. *Computer Science - Research and Development*, 29(3-4), 2014. DOI: 10.1007/s00450-013-0243-7.
12. Cristina Silvano, Giovanni Agosta, Stefano Cherubin, Davide Gadioli, Gianluca Palermo, Andrea Bartolini, Luca Benini, Jan Martinovič, Martin Palkovič, Kateřina Slaninová, João Bispo, João M. P. Cardoso, Rui Abreu, Pedro Pinto, Carlo Cavazzoni, Nico Sanna, Andrea R. Beccari, Radim Cmar, and Erven Rohou. The antarex approach to autotuning and adaptivity for energy efficient hpc systems. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '16, pages 288–293, New York, NY, USA, 2016. ACM.
13. Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009.

Acknowledgements

This work was supported by the PRACE project funded in part by the EUs Horizon 2020 research and innovation programme (2014-2020) under grant agreement 653838.