Domain Knowledge Specification for Energy Tuning

Anamika Chowdhury[1]
Madhura Kumaraswamy, Michael Gerndt[2]
Zakaria Bendifallah, Othman Bouizi[3]
Lubomír Říha, Ondřej Vysocký, Martin Beseda, Jan Zapletal[4]

The European Horizon 2020 project READEX is developing a tool suite for dynamic energy tuning of HPC applications. While the tool suite supports an automatic approach, domain knowledge can significantly help in the analysis and the runtime tuning phase. This paper presents the means available in READEX for the application expert to provide his expert knowledge to the tool suite.

1 Introduction

Energy efficiency and consumption have become the most important and challenging issues in current HPC systems and in designing future exascale computing systems. Advances in hardware technology, the operating and tuning HPC applications are required to reduce the overall energy consumption. Aspects such as arithmetic intensity and resource utilization can be exploited to benefit energy savings due to their ability to characterize varying application behaviour. In previous works, the energy consumption was optimized by using a model-based approach to predict and statically set the best frequency for the entire application run. However, in READEX, we develop a tool suite that switches tuning parameters dynamically during the application execution based on the dynamic changes over runtime [3].

The READEX methodology is a two-stage approach and consists of Design Time Analysis (DTA) and Runtime Application Tuning (RAT). It uses the Periscope Tuning Framework (PTF) [1] for DTA, and the READEX Runtime Library (RRL) for runtime tuning, with Score-P [2] as the common instrumentation and measurement infrastructure. Pre-analysis steps are performed prior to the DTA in which the application is instrumented and analyzed for dynamism. Coarse granular program regions that constitute most of the execution time are selected for dynamic tuning and are identified as *significant regions* [5].

A novel tuning plugin was developed for PTF to perform DTA and run experiments that currently evaluate three tuning parameters: CPU frequency, uncore frequency and the number of OpenMP threads within a single program run. The experiments are executions of the so-called *phase region*, which is usually the body of the main progress loop and whose individual time steps are called *phases*. The plugin then determines the best *configuration* or settings of the tuning parameters for the *runtime situations* (rts's) of significant regions, i.e., its instances at runtime. Rts's that have similar characteristics are grouped into

[1]Technical University of Munich, Faculty of Informatics, Germany, chowdhua@in.tum.de

[2]Technical University of Munich, Faculty of Informatics, Germany, kumarasw@in.tum.de

[3]Intel ExaScale Labs, Paris, France, zakaria.bendifallah@intel.com

[4]IT4Innovations National Supercomputing Centre, VŠB-TUO, Ostrava, Czech Republic, lubomir.riha@vsb.cz, ondrej.vysocky@vsb.cz, martin.beseda@vsb.cz, jan.zapletal@vsb.cz

scenarios, and best configurations for those scenarios are set. The knowledge obtained during DTA, such as the best-found system configurations for individual scenarios is encapsulated in a *tuning model*. For production runs, this tuning model is forwarded to the READEX Runtime Library (RRL), which performs runtime tuning by dynamically switching to the best configurations for upcoming rts's.

READEX uses the so-called *identifiers* to predict at runtime the characteristics of an upcoming rts by letting the developer specify domain knowledge. Currently, READEX supports region identifiers to distinguish rts's, phase identifiers to distinguish phase characteristics and input identifiers to distinguish executions with different application inputs. Without these identifiers, rts's of a significant region may be merged into the same scenario even if they have different behaviour. Hence, these identifiers will improve the tuning model by distinguishing rts's and assigning them to different scenarios to potentially select a better configuration. The domain knowledge also includes Application-level Tuning Parameters (ATP) that switch the application control flow and expose tuning potential in the target application.

This paper describes how the domain knowledge is used by the READEX tuning plugin during DTA and in brief, the RAT.

Listing 1: Phase specification

```
#include "SCOREP_User.inc"

SCOREP_USER_REGION_DEFINE(R1)


do it=1,max_iter
! phase region begins
  SCOREP_OA_PHASE_BEGIN(R1,...)
    ...
    call VCycle(...)
    ...
  SCOREP_OA_PHASE_END(R1)
! phase region ends
enddo
```

Listing 2: Region identifiers

```
!--- VCycle ---!
...
!--- level k-1 to level k ---!
do  k = min_level+1,max_level
  call interpolate(...,k)
  call resid(...)
  call psinv(...)
enddo

!Interpolate to level k region
subroutine interpolate(...,k)
  SCOREP_USER_PARAMETER("level",k)
  ...
end subroutine
```

2 Domain Knowledge Specification

This section describes how the user can define the Score-P Online Access Phase, provide additional identifiers, and specify application-level tuning parameters. Listings 1 and 2 show a high-level view of the domain knowledge specification for the MG (MultiGrid) benchmark of the NAS parallel benchmark suite. MG uses a V-cycle to solve a discrete Poisson equation on a 3D grid. It is based on a hierarchy of grid levels, where the maximum level is the finest grid with the highest resolution.

During each iteration, an entire V-cycle is executed starting from the highest grid level. The residual on the current grid level k is projected to the next coarser grid level *k-1*. When the coarsest grid is reached, an approximate solution is computed. The result is then interpolated from the coarser to the finer grid, where the residual is calculated and a smoother is applied to correct the result. The result is then propagated further upwards.

2.1 Phase Specification

Before starting DTA, the user must annotate the phase region with Score-P macros. It must first be declared, as shown in line 3 in Listing 1, and then surrounded by begin and end marcros as shown in lines 8 and 12 respectively in Listing 1. The Score-P user manual provides more information on the parameters of the macros.

2.2 Identifier Specification

The READEX tool suite provides support for different types of identifiers for runtime situations.

Region identifiers: The user can specify region identifiers via Score-P user parameters to distinguish rts's of that region if the region has different characteristics in the runtime situations. For example, since the size of the grid processed in *interpolate(...,k)* gets larger when going from the minimum grid level to the maximum, at a certain grid level the computation switches from being compute bound to memory bound. To enable DTA to determine special system configurations for compute and memory bound rts's, a region identifier for the grid level is added to the code (Line 12 of Listing 2). The region name, the call path, and the region identifier are now used as identifiers of the different rts's.

Phase identifiers: DTA also exploits dynamicity in the characteristics of the application across phases. To do so, the application expert can provide phase identifiers as domain knowledge at the start of the phase via region identifiers for the phase region.

Input identifiers: READEX also improves the tuning model by identifying special system configurations for different inputs characteristics. For example, in the multigrid application, the grid level where the computation switches from compute to memory bound depends on the resolution of the finest grid and the number of MPI processes. The finer the grid, the more levels are memory bound. The more processes are used, the fewer levels are memory bound due to an increased amount of cache. Application specific input identifiers are specified in an accompanying input specification file in the form of key-value pairs. These specification files will be used by both PTF and RRL. The number of MPI processes and OMP threads will be known implicitly.

2.3 Application Tuning Parameters

In order to leverage application dynamism, READEX enables to exploit the dynamism available through the use of different code paths such as the use of different preconditioners in the ESPRESO FEM library or different blocking factors in stencil codes.

Part of the READEX tool-suite, the *ATP library* provides an API to annotate the source code in order to identify the control variables responsible for control flow switching. During the first phase of the application execution in DTA, variable types, value ranges and addresses are discovered and agglomerated into an *ATP description file*. The subsequent phases of the application execution are reserved for best configuration discovery. Parameter information collected in the ATP description file is exploited to test different parameter values.

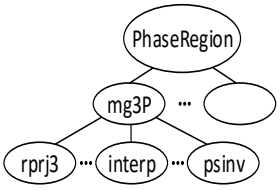

Figure 1: CCT of MG without user parameters

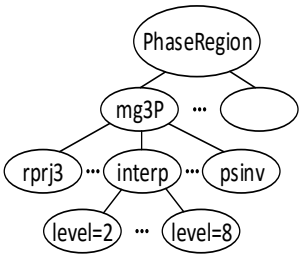

Figure 2: CCT of MG with user parameters

Furthermore, one critical complexity that the exploitation of ATPs exhibits is the presence of dependences between variables, where the values for one variable depend on the values of prior variable(s). In this case, not all value combinations are possible and forcing the values on the program may break its semantics. READEX, through the ATP library, provides the means to handle this by allowing the declaration of parameter dependences in the form of logical constraints. It also relies on a constraints solver called *the omega library*[5] to resolve the dependences. The solver handles affine function based constraints and provides valid combinations of parameter values for use in the DTA phase.

3 Implementation

DTA is carried out by PTF, a distributed framework consisting of a frontend and a hierarchy of analysis agents [1]. First, one phase of the application is executed in which the application regions are gathered and returned to the PTF analysis agents from Score-P via the Online-Access Interface and to generate the ATP specification file. Partial *Calling Context Trees* (CCT) [6] are generated at every analysis agent for those MPI processes controlled by the agent, and are gathered to create the complete tree in the frontend.

Figure 1 presents the CCT of MG if no region identifier is given. Separate nodes are created for the call sites of the projection, interpolation, and the smoother. All runtime situations, i.e. invocations, are represented by one node with its call path. Figure 2 illustrates the situation with the region identifier in the interpolation. For each grid level, a separate node is created and the runtime situations can be distinguished in PTF. Each rts is identified by its region name, the call path, which includes the region identifiers (represented as *parameter_name=value*), and the phase and input identifiers. With the region identifier, a valid rts of the region `interp` is */PhaseRegion/mg3P/interp/level=8*, as shown in Figure 2.

The PTF frontend executes the READEX tuning plugin, which reads from the READEX configuration file the objective(s) (Energy, CPU Energy, Execution Time, Energy Delay Product or Energy Delay Product Squared), the tuning parameters (core frequency, uncore frequency and the number of OpenMP threads), the search strategy (exhaustive, random, individual or genetic) and the significant regions. It also reads the input identifiers from the input specification file and the ATPs from the generated ATP specification file.

[5] http://www.cs.umd.edu/projects/omega/

[6] A context sensitive version of a call graph.

It then assesses selected system configurations from the search space of the tuning parameters and the ATPs generated by the search algorithm. For each configuration, it executes an experiment and measures the objective value for all the rts's of the significant regions. The plugin outputs the best configuration for both the phase and the rts's.

Finally, a *tuning model* is generated from this knowledge. The tuning model generation clusters the rts's into *scenarios* based on their best configuration. It determines a *classifier* that maps each valid rts onto a unique scenario based on the identifiers given at runtime. For each scenario, a *selector* is generated that returns a single or a set of good configurations for that scenario with respect to the chosen objective. The tuning model encapsulates this knowledge, and is stored as a JSON file, which is then read by the RRL to perform dynamic switching at runtime.

4 Example

The ESPRESO [4] library is a combination of Finite Element (FEM) tools and a domain decomposition based Finite Element Tearing and Interconnect (FETI) solvers. The FETI solver contains a projected conjugate gradient (PCG) solver and therefore, its convergence can be improved by several preconditioners. The computational complexity of different preconditioners vary from basic vector scaling (weight function), to sparse-matrix vector multiplication with different number of non-zeros (lumped, light-dirichlet) to dense matrix-vector multiplication (dirichlet). Using a simplified approximation, we can state that from the preconditioners listed above, the more computationally demanding the preconditioner is, the more numerically efficient it is, i.e. the more it reduces the number of iterations to solve the problem. In ESPRESO, we can dynamically switch between any of these during the runtime. If a preconditioner is not used, one iteration contains an action of a FETI operator (cost is 30.9 J and 0.12 s) and an application of a projector (cost is 0.7 J and 0.005 s). If a preconditioner is used, each iteration contains one more projector application in addition to the preconditioner action.

We evaluated the preconditioners on a structural mechanics (linear elasticity) problem with 2.3 million unknowns on a single compute node using 24 MPI processes. The results, see Table 1, show that the solution can be reached in 5.46 s when using Light Dirichlet preconditioner, despite the fact that it needs more iterations than the Dirichlet preconditioner. The Light Dirichlet preconditioner saved 15.9 s and 4 091.5 J in comparison to solving the problem without any preconditioner.

Preconditioner	**# iterations**	**1 iteration**		**Solution**	
none	172	125 ms	31.6 J	21.36 s	5 501.31 J
Weight function	100	130+2 ms	32.3+0.53 J	12.89 s	3 284.07 J
Lumped	45	130+10 ms	32.3+3.86 J	6.32 s	1 636.11 J
Light dirichlet	39	130+10 ms	32.3+3.74 J	5.46 s	1 409.82 J
Dirichlet	30	130+80 ms	32.3+20.62 J	6.34 s	1 594.50 J

Table 1: ESPRESO preconditioners comparison for runtime and energy consumption. The table contains (i) single iteration evaluation including baseline (FETI operator and 2x projector) + resources spent by the preconditioner (ii) overall FETI solver evaluation considering the different number of solver iterations.

5 Conclusion

This paper gave a short overview of the READEX project, which is aiming at improving the energy efficiency of HPC applications by a dynamic tuning approach. At design time, a tuning model that guides the dynamic switching of tuning parameters is determined. The quality of that tuning model can be enhanced by domain knowledge that is provided by the application owner. Part of the domain knowledge are application-level tuning parameters that significantly increase the tuning potential. The tuning potential is demonstrated in an example, where ATPs are used to select different preconditioners for the ESPRESO library.

References

[1] M. Gerndt, E. César, and S. Benkner, eds., *Automatic Tuning of HPC Applications - The Periscope Tuning Framework*, Shaker Verlag, Aachen, 2015.

[2] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. D. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, *Score-p: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir*, in Tools for High Performance Computing 2011, H. Brunst, M. Müller, W. E. Nagel, and M. M. Resch, eds., Springer, Berlin, 2012, pp. 79–91.

[3] Y. Oleynik, M. Gerndt, J. Schuchart, P. G. Kjeldsberg, and W. E. Nagel, *Run-time exploitation of application dynamism for energy-efficient exascale computing (READEX)*, in Computational Science and Engineering (CSE), 2015 IEEE 18th International Conference on, C. Plessl, D. El Baz, G. Cong, J. M. P. Cardoso, L. Veiga, and T. Rauber, eds., Piscataway, Oct 2015, IEEE, pp. 347–350.

[4] L. Riha, T. Brzobohaty, A. Markopoulos, O. Meca, and T. 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.

[5] J. Schuchart, M. Gerndt, P. G. Kjeldsberg, M. Lysaght, D. Horák, L. Říha, A. Gocht, M. Sourouri, M. Kumaraswamy, A. Chowdhury, M. Jahre, K. Diethelm, O. Bouizi, U. S. Mian, J. Kružík, R. Sojka, M. Beseda, V. Kannan, Z. Bendifallah, D. Hackenberg, and W. E. Nagel, *The READEX formalism for automatic tuning for energy efficiency*, Computing, (2017), pp. 1–9. DOI: 10.1007/s00607-016-0532-7.

Acknowledgements

The research leading to these results has received funding from the European Union's Horizon 2020 Programme under grant agreement number 671657.