DSL and Autotuning Tools for Code Optimisation on HPC Inspired by Navigation Use Case

Jan Martinovič, Kateřina Slaninová and Martin Golasowski
IT4Innovations, VŠB — Technical University of Ostrava

Radim Cmar
Sygic

Joao M. P. Cardoso, and Joao Bispo
FEUP, University of Porto

Gianluca Palermo, Davide Gadioli, and Cristina Silvano
DEIB, Politecnico di Milano

IT4Innovations national supercomputing center

Sygic

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

POLITECNICO MILANO 1863

Instrumentation via DSL

LARA ASPECT

```
aspectdef ObserveScalability {
   input fname, inc, max_threads, runs end
   select function.call{name==fname} end
   apply
   insert.before  %{‘
   for (int threads = 1; threads <= [[max_threads]];
threads += [[inc]]) {
            omp_set_num_threads(threads);
            std::vector<long> times([[runs]]);
            std::vector<int> results([[runs]]);
            for(int r = 0; r < runs; r++) {
      }%
    insert.replace ‘results[r]  =
[[$call.statement]]’;
    insert.after  %{‘
      }
      int sum = 0;
      for (int r = 0; r < [[runs]]; ++r)
      {
        sum += times[r];
      }
      float average = (sum/runs);
    }%
  end
```

LARA ASPECT

```
aspectdef MeasureExecTime {
   input fname, store,  end
   select function.call{name==fname} end
   apply
      insert.before %{auto startTime =
std::chrono::high_resolution_clock::now();}%;
      if(store != undefined) {
        insert.after ‘[[store]] =
std::chrono::duration_cast<std::chrono::millisecon
ds>(std::chrono::high_resolution_clock::now() -
startTime).count();’;
      } else {
        insert.after
‘std::chrono::duration_cast<std::chrono::milliseco
nds>(std::chrono::high_resolution_clock::now() -
startTime).count();’;
}
   end
}
```

```
aspectdef MeasureScalability
{
   Call MeasureExecTime(“times[r]”);
   Call ObserveScalability
(“RunMonteCarloAll”, 1, 16, 100);
}
```

The domain specific language (DSL) and toolflow approach proposed in the ANTAREX project can provide the mechanisms needed for addressing properly the problems previously described. The DSL being developed is based on the LARA DSL and allows to specify strategies for code instrumentation and code transformations, including the required code adaptation for dynamic autotuning.

INSTRUMENTED SOURCE CODE

ObserveScalability

MeasureExecTime

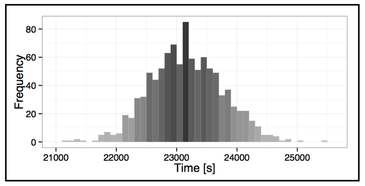
```
for (int threads = 1; threads <= max_threads; threads += increment) {
    omp_set_num_threads(threads);
    std::vector<long> times(runs);
    std::vector<int> results(runs);
    for(int r = 0; r < runs; r++) {
      auto startTime = std::chrono::high_resolution_clock::now();
      results[r] = RunMonteCarloAll(segments, samples)[0];
      times[r] = std::chrono::duration_cast<std::chrono::milliseconds>
(std::chrono::high_resolution_clock::now() - startTime).count();
    }
    int sum = 0;
    for (int r = 0; r < runs; ++r) {
      sum += times[r];
    }
    float average = (sum/runs);
 }
```

Self-adaptive Navigation System

Our system is a combination of **server-side** and **client-side** navigation which aims to provide the **most efficient navigation** in the context of **smart cities**. The routes are provided with regard to a global optimum for a city. This is a **complex task** which requires the computational **power of the HPC infrastructure**. Efficient operation of the system is supported by the **custom domain-specific language (DSL) LARA** and the **autotuning framework** being developed within the **ANTAREX project**.

Probabilistic Time-Dependent Travel Time Computation algorithm has been selected for the demonstration of the DSL and autotuning framework. The input for the algorithm is a departure time for a selected route represented as a line of road segments. The algorithm provides estimated probability distribution of the travel time along the specified route.

The computation is based on the Monte Carlo simulation. It randomly selects probability speed profiles on road segments and computes travel time at the end of the route. Number of random samples greatly affects precision of the result. Many simulations have to be executed in order to satisfy demand for the precise results of the simulation in the context of the smart city.



ANTAREX 10^18

AutoTuning and Adaptivity appRoach for Energy efficient eXascale HPC systems

Scalability measurement of the simulation code instrumented by the DSL

No manual code changes are required. Performance of the code under different execution conditions is measured automatically by combination of several LARA aspects. The conditions are defined by several parameters:

- Number of OpenMP threads
- Thread scheduling strategy (compact, scatter)
- Cache usage strategy

Autotuning Integration

INSTRUMENTED CODE

```
std::vector<float> RunMonteCarloAll(int samples, int secs,) {
...
   #pragma omp parallel for
   for (int s = 0; s < samples; ++s) {
     for (int d = 0; d < 7; ++d) {
       for (int i = 0; i < intervalsPerDay; ++i) {
         viRngUniform(VSL_RNG_METH_UNIF_STD, rndStream, probsSize,
probs,0, INDEX_RESOL);
         travelTimes[(d * intervalsPerDay * samples) + (i * samples) +
s] = GetRandomTravelTime(secs, probs);
       }
     }
   }
...
```

INSERT OPENMP PRAGMA

```
aspectdef OMPParFors {
    input fname end
    select function{name==fname}.loop end
    apply
    if($loop.is_parfor && $loop.is_outermost) {
        insert.before ‘#pragma omp parallel for’;
    }
  end
}
```

Integration of mechanisms and strategies to on-line adapt the application behaviour and the platform configuration with respect to changing workloads, operating conditions and computing resources.

INSTRUMENTED

```
...
omp_set_dynamic(0);
Margo::MC::update(&threads);
omp_set_num_threads(threads);
Margo::MC::start_monitor();
RunMonteCarloAll(segments, samples);
Margo::MC::end_monitor();
...
```

EXPOSE SOFTWARE KNOBS

```
aspectdef OMPCtrThreads {
   input fname, knob=“threads” end
   select function.call{name==fname} end
   apply
      insert.before %{omp_set_dynamic(0);
          omp_set_num_threads([[$knob]]);}%;
   }
  end
}
```

- Searching the best combination of the software knobs impacting the given metrics (for example execution time, energy consumption, etc.)
- Definition of the goals to drive the optimization of the execution environment (for example to comply with the target SLA)

INTEGRATE AUTOTUNER

```
aspectdef AutoTuneThreads {
   input fname end
   select function.call{name==“omp_set_num_threads”} end
   apply
      insert.before ‘Margo::MC::update([[$call.arg]]);’;
   end

   select function{name==fname} end
   apply
      insert.before ‘Margo::MC::start_monitor();’;
      insert.after ‘Margo::MC::end_monitor();’;
   end
}
```

AUTOTUNING

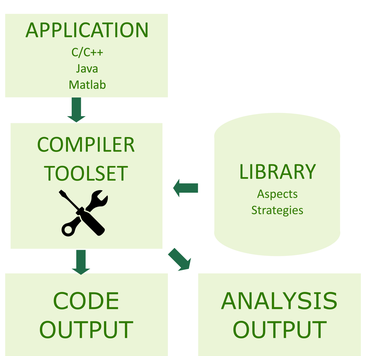
GOALS AND PARAMETERS

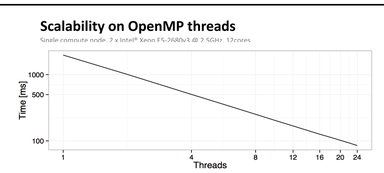
```
<!-- SW-KNOB SECTION -->
<knob name="num_threads" var_name="threads" var_type="int"/>

<!-- GOAL SECTION -->
<goal name="MC_responseTime_goal"
monitor="MC_responseTime_monitor" dFun="Average" cFun="LT" value="MC_SLA" />

<!-- OPTIMIZATION SECTION -->
<state name="power_optimised" starting="yes" >
 <minimize combination="linear">
  <metric name="MC_avg_power" coef="1.0"/>
 </minimize>
 <subject to="MC_responseTime" metric_name="MC_responseTime" priority="1" />
</state>
```

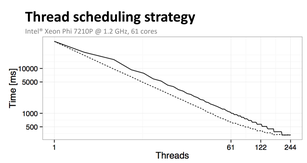
Toolset workflow

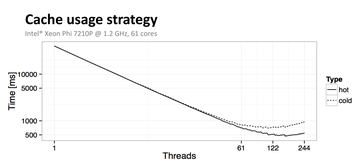
Apart from the instrumentation, the aspects can be used to define the optimization logic which would select the best environment parameters (knobs) according to selected metric (speedup, energy consumption, etc.).

Library of different instrumentation strategies can be created, providing various useful analyses of the code performance. The main advantage of this approach is separation of the strategies (including instrumentation and code transformations) from the actual source code. The strategies can be programmed as generic and applicable to wide range of different source codes.









ANTAREX Project Goals

The main goal of the ANTAREX project is to provide a breakthrough approach to express by a Domain Specific Language the application self-adaptivity and to runtime manage and autotune applications for green and heterogeneous High Performance Computing (HPC) systems up to the Exascale level.

- **Dynamic self-monitoring** and **self-adaptivity** or **«autotuning»** HPC applications with respect to changing workloads, operating conditions and computing resources
- **Programming models and languages to express self-adaptivity and extra-functional properties**
  - ⇒ Introducing a separation of concerns between extra-functional strategies (self-adaptivity, parallelisation, energy/thermal management) and application functionality by the design of a new aspect-oriented Domain Specific Language
- **Exploiting heterogeneous computing resources in Green HPC platforms by runtime resource and power management**