How to Scale Dynamic Tuning to Large Parallel Applications

Andrea Martínez, Anna Sikora, Eduardo César, Joan Sorribes
Outline

• Motivation.

• Large-scale Dynamic Tuning.

• Scalability Evaluation.

• Conclusions and Future Work.
Motivation

Number of Cores in Top 10 Supercomputers

Year

- 1996
- 2000
- 2004
- 2008
- 2012

Performance?

Performance Analysis Tools

CHALLENGES

Dynamic tuning

- Active Harmony
- MATE

Scalability

- Periscope
- Scalasca

Computer Architecture and Operating Systems Department
Universidad Autónoma de Barcelona
Motivation

Define an approach that provides automatic and dynamic analysis and tuning of large-scale parallel applications.
Motivation

Define an approach that provides automatic and dynamic analysis and tuning of large-scale parallel applications.
Outline

• Motivation.

• Large-scale Dynamic Tuning.

• Scalability Evaluation.

• Conclusions and future work.
Hierarchical Tuning Network

• It is composed of **analysis and tuning modules (Analyzers)** structured as a hierarchical tree.

• The base of the hierarchy is composed of Analyzers that controls disjoint subsets of application tasks.

• Abstraction mechanism between levels of Analyzers.

---

**Monitoring**

An analyzer must be able to present itself, to its parent Analyzer, as a parallel application task ...

**Analysis**

...and abstracts the behavior of its analysis and tuning domain.

**Tuning**

Analysis and tuning domain
Abstraction mechanism

- Abstraction model
  - How to translate instrumentation orders to be applied to the child level.
  - How to create new events which satisfy the requests of previous monitoring orders.

- Application task
  - Analyzer
    - Performance model
      - Monitoring points
      - Performance functions
      - Tuning points

- User
  - Instrumentation order sender
  - Internal API

Instrumentation orders ➔ Events ➔ Level i+1 ➔ Level i ➔ User
Abstraction example

A1 generates:

- **Mon_Order**(computation_time, #work_units) to A1.1
- **Mon_Order**(computation_time, #work_units) to A1.2
- **Mon_Order**(computation_time, #work_units) to A1.3
- **Mon_Order**(computation_time, #work_units) to A1.4

Abstractor associated to A1.1 generates:

- **Mon_Order**(computation_time, #work_units) to T1
- **Mon_Order**(computation_time, #work_units) to T2
- **Mon_Order**(computation_time, #work_units) to T5
- **Mon_Order**(computation_time, #work_units) to T6.
Abstraction example

A1.1 receives:

- \textbf{Event}(\text{computation\_time}, \#\text{work\_units}) from T1
- \textbf{Event}(\text{computation\_time}, \#\text{work\_units}) from T2
- \textbf{Event}(\text{computation\_time}, \#\text{work\_units}) from T5
- \textbf{Event}(\text{computation\_time}, \#\text{work\_units}) from T6

Using the abstraction model, the Abstractor associated to A1.1 creates a new event:

- \textbf{Event}(\text{Mean}(\text{computation\_time}), \text{Sum}(\#\text{work\_units})) to A1
A1 generates:

- `TuningOrder(send_load, to_A1.3, #work_units)` to A1.1

Using the abstraction model, the Abstractor associated to A1.1 generates:

- `TuningOrder(send_load, to_T9, #work_units/2)` to T5
- `TuningOrder(send_load, to_T10, #work_units/2)` to T6
Determining the architecture topology

The use of architecture topologies with the minimum number of non-saturated analysis modules.

Work during the analysis and tuning process

\[ N \cdot E_a \cdot T_m + T_a(N) + \frac{E_a}{E_c} \cdot T_c + T_t \cdot f_{rp} \]

- Gathering performance data
- Performance analysis
- Event creation
- Order translation

Parallel application of N tasks

Parallel application of N tasks

Computer Architecture and Operating Systems Department
Universidad Autónoma de Barcelona
Outline

• Motivation.

• Large-scale Dynamic Tuning.

• Scalability evaluation.

• Conclusions and future work.
Scalability Evaluation

- Prototype implementation of the proposed tuning network

**MRNet Framework**

- A network of hierarchically organised processes.
- Potential: Filters.

The prototype simulates all the actions which would take place during the performance analysis and tuning process of a parallel application:

- BEs simulate an instrumented SPMD application.
- Filters simulate the Abstractor-Analyzer functionality.
Scalability Evaluation

• Execution Environments:
  – Marenostrum at Barcelona Supercomputing Centre
  – SuperMUC at Leibniz Supercomputing Centre

<table>
<thead>
<tr>
<th># Tasks of the parallel application</th>
<th>Level 0</th>
<th>Level 1</th>
<th>Level 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Analysis modules</td>
<td>Domain size</td>
<td>#Analysis modules</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>5</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>19</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td>800</td>
<td>37</td>
<td>22</td>
<td>2</td>
</tr>
</tbody>
</table>

Architectures executed in Marenostrum and SuperMUC

| 1600 | 73 | 22 | 4 | 20 | 1 | 4 |
| 3200 | 146 | 22 | 7 | 21 | 1 | 7 |
| 6400 | 292 | 22 | 14 | 21 | 1 | 14 |
Scalability Evaluation

Time to make a decision

- Network transfer
- Management
- Performance analysis
- New event creation

Decision time at level 0
Decision time at level 1
Global decision time
Scalability Evaluation

![Scalability Graph]

Scalability at MareNostrum

- Level 0
- Level 1
- Level 2

Scalability at SuperMUC

- Level 0
- Level 1
- Level 2

Time to make a decision (ms)

Number of tasks in the parallel application

Computer Architecture and Operating Systems Department
Universidad Autónoma de Barcelona
Scalability Evaluation

Efficiency of the hierarchical tuning network

When is an analysis module saturated?

Scalability

- Event batch
- Management
- Analysis

Analysis lag

$N \cdot E_a \cdot T_m + T_a(N) + \frac{E_a}{E_c} \cdot T_c + T \cdot f_{rp}$

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Level 0</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Analysis modules</td>
<td>Domain size</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>22</td>
</tr>
</tbody>
</table>

Computer Architecture and Operating Systems Department
Universidad Autónoma de Barcelona
Scalability Evaluation

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Level 0</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Analysis modules</td>
<td>Domain size</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>22</td>
</tr>
</tbody>
</table>

Analysis lag of level 0 analysis modules

- **Saturation state**
- **Stable state**

Experiment 7 will be greatly degraded by the saturation of the analysis module due to having fewer level 2 analysis modules than necessary. It presents an analysis lag which steadily increases during the experiment’s execution. This is because the analysis module is unable to complete the analysis and tuning process carried out by the level 3 analysis module. However, this effect is not seen in Experiment 3 and 4, as demonstrated by the analysis lag results for these experiments.

The analysis lag is directly proportional to the number of nodes within the analysis and tuning domain. In this case, all three experiments show analysis lag which steadily increases during the experiment time.

Current effort in Periscope is the best candidate to work on large scale systems, managing the scalability of parallel applications. Autopilot supports dynamic tuning in heterogeneous environments. There are also other tools which perform dynamic tuning.

For this purpose, we have chosen the topology of analysis modules composed of a fewer and greater number of analysis modules size. The topology is built using expression 4 with topologies of analysis lag from Experiment 7 that reflects its saturation due to being unable to complete the analysis and tuning process within the analysis period.

For these 7 experiments, the analysis lag is calculated for 822 parallel application tasks. The results show the expected behavior wherein the analysis lag is directly proportional to the number of children far below their maximum domain size.
Scalability Evaluation

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Level 0</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Analysis modules</td>
<td>Domain size</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>22</td>
</tr>
</tbody>
</table>

![Analysis lag of level 1 analysis modules]

Analysis lag (ms)

Stable state
Outline

• Motivation.

• Large-scale Dynamic Tuning.

• Scalability Evaluation.

• Conclusions and future work.
Conclusions

• A model for distributed performance analysis based on a hierarchical tuning network has been defined.

• The decentralised decision making process employs user provided performance models and an abstraction mechanism.

• The scalability of the proposed model has been verified using a prototype of the tuning network.

Future work

• Show the benefits of our approach when applied to real large-scale applications in order to improve their performance.

• Combine our approach with the one implemented under the AutoTune project.
Thank you for your attention

amartinez@caos.uab.es

Andrea Martínez, Anna Sikora, Eduardo César, Joan Sorribes

Computer Architecture and Operating Systems Department
Universidad Autónoma de Barcelona
Determining the architecture topology

Parallel application of N tasks

The use of architecture topologies with the minimum number of non-saturated analysis modules.

Work during the analysis and tuning process

\[ N \cdot E_a \cdot T_m + T_a(N) + \frac{E_a}{E_c} \cdot T_c + T_t \cdot f_{rp} \]

- Gathering performance data
- Performance analysis
- Event creation
- Order translation

256 tasks
Parallel application

Computing Architecture and Operating Systems Department
Universidad Autónoma de Barcelona
Hierarchical Tuning Network

Hierarchical Master/Worker

Master/Worker of pipelines