An Automatic Design Space Exploration Framework for Multicore Architecture Optimizations

Horia Calborean, PhD student

Prof. Lucian Vinţan, PhD supervisor

Outline

- Multiobjective optimization
- Related work
- The developed framework
  - XML interface
  - Implemented algorithms
  - Implemented metrics
  - Implemented test functions
- Simulation results
- Conclusions and further work

Multiobjective optimization

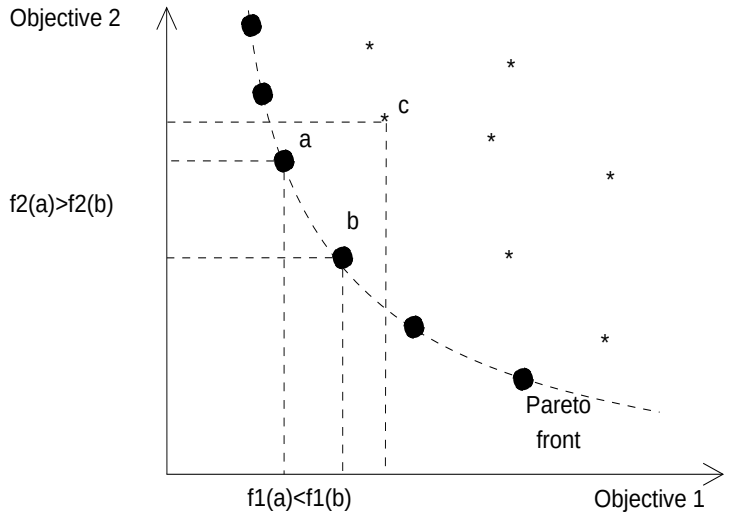
- The number of (heterogeneous) cores integrated in the processor, has risen to tens, hundreds or even thousands (GPUs)
- As the number of cores becomes higher, more configurations have to be simulated
- This leads to an extremely huge search space (NP-hard). The current processor optimization methodology will not scale and new methods are needed.
- Performance evaluation has become a complex multiobjective evaluation (speed , power consumption, area integration, etc.)

A multiobjective optimization taxonomy

- Aggregating approaches
  - combine (or aggregate) all the objectives of the problem into one single objective
- Lexicographic ordering
  - user (decision maker) has to rank all the objectives in order of their importance
- Sub-Population approaches
  - several instances of a single objective algorithm run in parallel and try to optimize one of the objectives
- Pareto-based approaches
  - These approaches use individual selection techniques based on Pareto dominance
- Other
  - Hybrid methods, or other methods that do not fall in the above categories

Multiobjective optimization and Pareto optimality

- An order must be established between individuals
- The concept of Pareto optimality and the notion of dominance is used
- Domination relation: no order can be established between points a and b (see figure) but both a and b dominate c



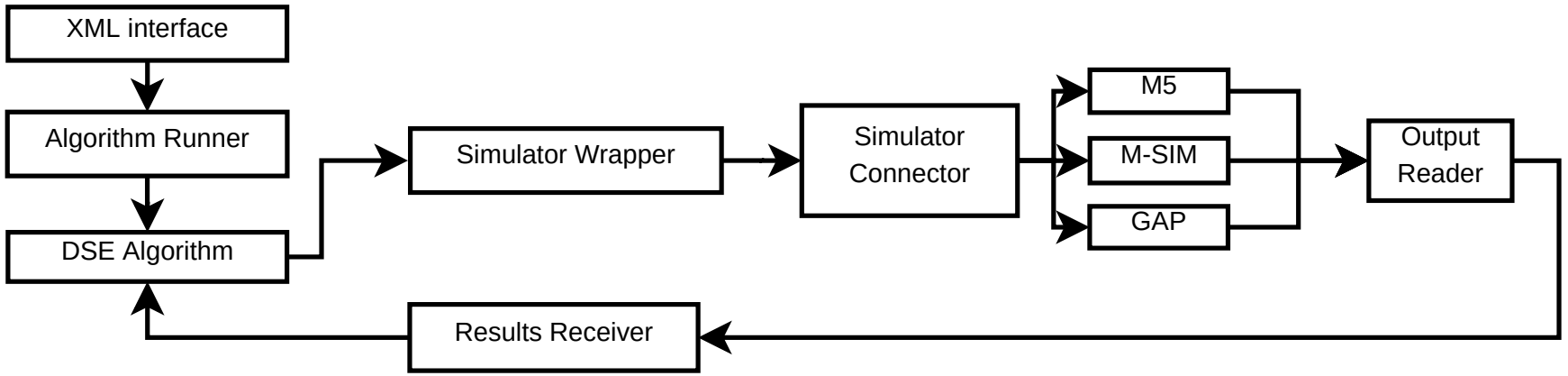
Related work

- M3Explorer
  - works only with simulators implemented in Linux
  - misses important genetic algorithms (SPEA2)
- Archexplorer
  - used to find best cache configuration
  - the user has to write an implementation of the cache which is compatible with the Archexplorer.
  - users can not change the algorithm (algorithm is not public)
- Magellan
  - bounded to one simulator. The user can not use his own simulator

Framework for Automatic Design Space Exploration (FADSE)

- Incorporates many multiobjective evolutionary algorithms (NSGA-II, SPEA2, PAES, etc.) through the integration with the jMetal library. We have already implemented two algorithms: SEMO and FEMO
- Users are able to use FADSE with almost any existing multicore or NoC simulator and on most platforms (implemented in JAVA)

Application structure



- After the XML input is loaded it is passed to the jMetal library which generates individuals. The individuals are sent to the (multicore) simulator.
- When the simulation is done the results are passed back to jMetal and the process restarts.

XML interface – simulator configuration

```
<simulator name="XSimulator" type="simulator" >
    <simulator_executable path="/path/to/simulator/executable/executable" />
    <simulator_output_file path="/path/to/output/file/out.txt"/>
</simulator>
<metaheuristic name="NSGA-II" config_path="nsgaii.properties" />
```

- Through the XML interface the user can configure FADSE to run a **simulator** or a **synthetic test problem**
- For the multicore simulator the executable and other parameters can be set
- The user will choose the desired DSE algorithm.

XML interface –parameters and objectives specification

```
<parameters>
    <parameter name="l1data_cache_size" description="" type="exp2" min="32" max="64"/>
    <parameter name="nr_of_cores" description="" type="integer" min="1" max="8"/>
    <parameter name="benchmark" description="param3" type="string">
        <item value="/benchmarks/fft"/>
        <item value="/benchmarks/ocean"/>
    </parameter>
</parameters>
<system_metrics>
    <system_metric name="IPC" type="integer" unit="" desired="big"/>
    <system_metric name="power_consumption" type="float" unit="" desired="small"/>
</system_metrics>
```

- Parameters (for the architecture and compiler) and their possible values have to be set
- There are multiple types of parameters (integer, float, list of strings, geometric progression)
- The objectives are specified and if they should be maximized or minimized

XML interface – rules – *Relational* rule

```
<rule name="minimum cache size">
    <greater-equal>
        <parameter name="l2_cache_size"/>
        <parameter name="l1data_cache_size"/>
    </greater-equal>
</rule>
```

- Constrains (rules) can be imposed
- Constraints are used to reduce the size of the search space and to develop (assure) valid individuals
- Valid relations: greater, greater and equal, less, less and equal, equal, not equal

XML interface – rules – *And* rule

```
<rule name="cache size check">
    <and>
        <greater-equal>
            <parameter name="l2_cache_size"/>
            <parameter name="l1data_cache_size"/>
        </greater-equal>
        <greater-equal>
            <parameter name="l2data_cache_size"/>
            <constant value="512"/>
        </greater-equal>
    </and>
</rule>
```

- The *And* rule is used when multiple rules have to be obeyed at the same time
- Any type and any number of rules can be put in an *And* rule

XML interface – rules – *If* rule

```
<rule name="cache associativity limitation">
    <if>
        <greater>
            <parameter name=" l2_cache_assoc "/>
            <constant value="1"/>
        </greater>
        <then>
            <equal>
                <parameter name=" l1_cache_assoc "/>
                <constant value="1"/>
            </equal>
        </then>
    </if>
</rule>
```

- The *If* rule: used when we want to impose a certain constraint only when another condition is met
- Any rule can be used in the condition and also inside the “then” clause

Implemented multiobjective algorithms: SEMO and FEMO

- SEMO and FEMO are genetic algorithms
- They use **only the mutation operator**
- **SEMO chooses randomly** an individual from the current population, mutates him and if it is non-dominated it inserts it in the population
- **FEMO** is similar with SEMO; the difference is that it **chooses the individual with the smallest number of offspring**.

Implemented metrics

- Error ratio
  - Measures the number of individuals in the Pareto optimal set that are not members of the Pareto front

$$ER = \frac{\sum_{i=1}^{|PF_{known}|} e_i}{|PF_{known}|}$$

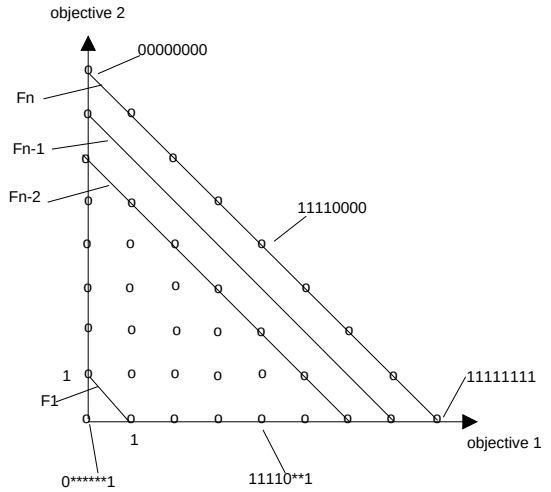
- Coverage of two sets
  - How many individuals from a population dominate individuals from another population

$$C(X', X'') = \frac{|\{a'' \in X''; \exists a' \in X' : a' \succeq a''\}|}{|X''|}$$

Implemented test functions: LOTZ and DTLZ family

- LOTZ

$$LOTZ(x_1,\ldots,x_n)=\left(\sum_{i=1}^{n}\prod_{j=1}^{i}x_j,\sum_{i=1}^{n}\prod_{j=i}^{n}(1-x_j)\right)$$



- DTLZ1

Minimize $\vec{f}(\vec{x})$

$$f_1(\vec{x})=\frac{1}{2}x_1x_2\ldots x_{m-1}(1+g(x_m,x_{m+1},\ldots,x_n)),$$

$$f_2(\vec{x})=\frac{1}{2}x_1x_2\ldots x_{m-2}(1-x_{m-1})(1+g(x_m,x_{m+1},\ldots,x_n)),$$

$$f_3(\vec{x})=\frac{1}{2}x_1x_2\ldots x_{m-3}(1-x_{m-2})(1+g(x_m,x_{m+1},\ldots,x_n)),$$

$$\ldots$$

$$f_{m-1}(\vec{x})=\frac{1}{2}x_1(1-x_2)(1+g(x_m,x_{m+1},\ldots,x_n)),$$

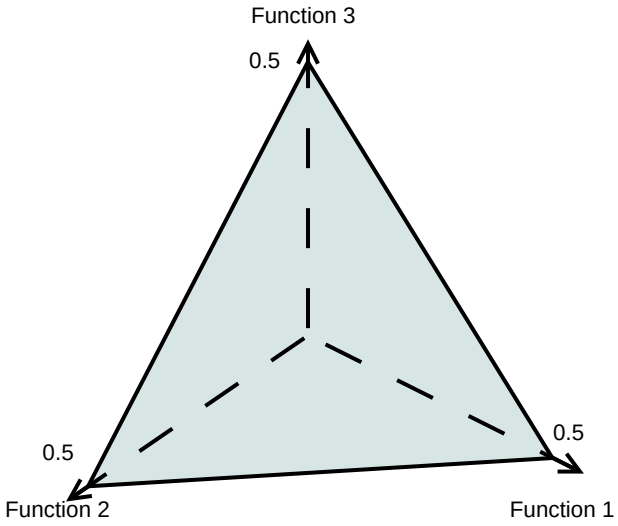
$$f_m(\vec{x})=\frac{1}{2}(1-x_1)(1+g(x_m,x_{m+1},\ldots,x_n)).$$

subject to $0\le x_i\le 1$ for $i=1,2,\ldots n$.

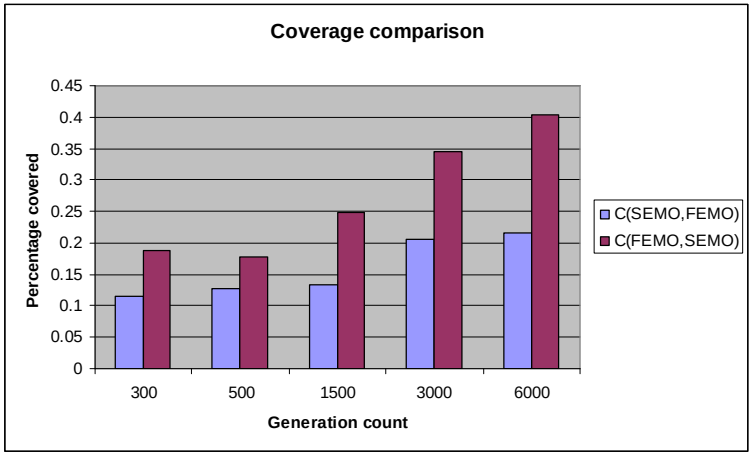
and

$$g(\vec{x_M})=100\left((n-m)+\sum_{i=m}^{n}(x_i-0.5)^2-\cos(20\pi(x_i-0.5))\right)$$

where $\vec{x_M}=(x_m,x_{m+1},\ldots,x_n)$.



Simulation results



Coverage comparisons between SEMO and FEMO on the DTLZ1 problem.

- LOTZ problem:
  - SEMO discovers the entire Pareto front in an average of 1453 generations. For FEMO the average number of generations was only 756.
- DTLZ1 problem
  - SEMO and FEMO are not able to reach the Pareto front (Error ratio was always 1 in our experiments (6000 generations, 1 offspring per generation) FEMO performs better than SEMO (see the above Figure)
- The implemented algorithms are able to solve the LOTZ problem with a fairly small amount of simulated individuals (1-2% from the total).

Conclusion and further work

- We have developed a framework which is able to perform automatic design space exploration
- It is easily extensible and portable
- We plan to integrate fully jMetal library to use the implemented algorithms
- FADSE will be a **client-server application** and the simulations will be done in parallel
- Integrate a database system to remember (reuse) already simulated individuals
- Write connectors to other simulators

Conclusion and further work

- Perform an evaluation of the existing DSE algorithms on different simulators
- Find out which one performs best (e.g. based on coverage metrics)
- Improve the DSE algorithms - map them on the specific problem of design space exploration

THANK YOU

You can contact me at:

horia.calborean@ulbsibiu.ro