Towards multi-objective, region-based auto-tuning for parallel programs

Thomas Fahringer

Juan Durillo, Philipp Gschwandtner, Herbert Jordan,
Klaus Kofler, Peter Thoman

distributed and parallel systems group
institute of computer science, university of Innsbruck

PPAM 2017, Sept 12, 2017, Lublin

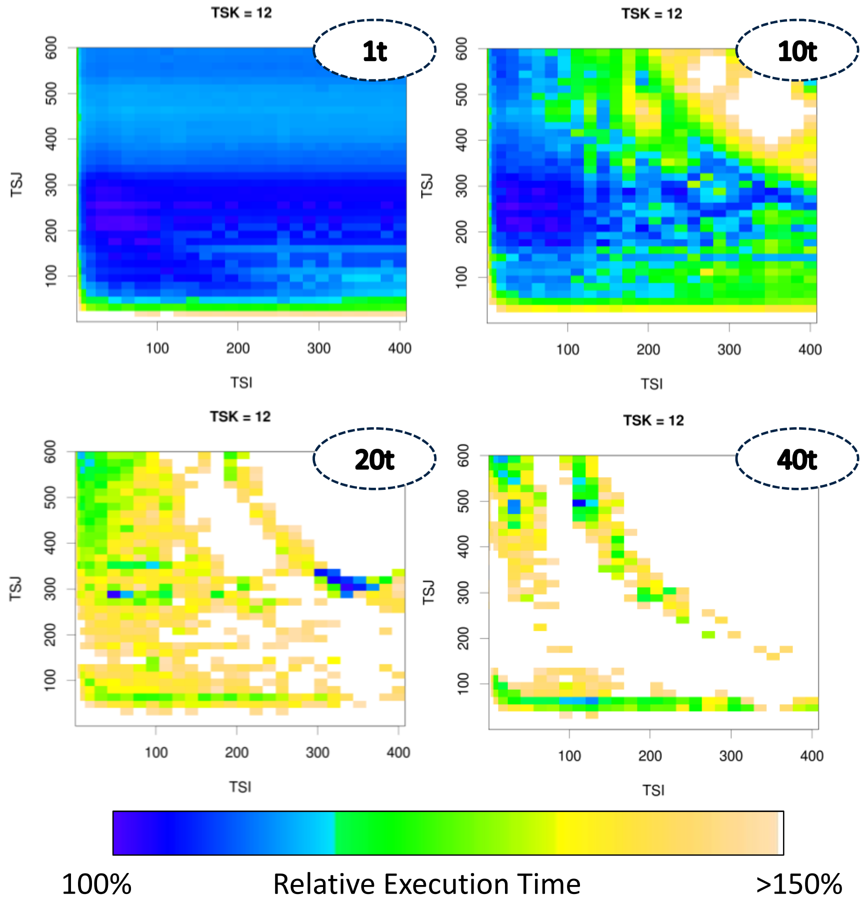
Why is it so hard to optimize codes for parallel systems?

- Question:
  - If the strategy for I/O scheduling, process scheduling, cache replacement policy would be changed, how would you re-write your code?
- Complexity, undecidability and difficulty to predict program and system behavior:
  - Dynamic re-allocation of cores and memory, clock speed, external load, sharing of resources, etc.
  - Operating system, external load, queuing systems, caches often difficult to predict

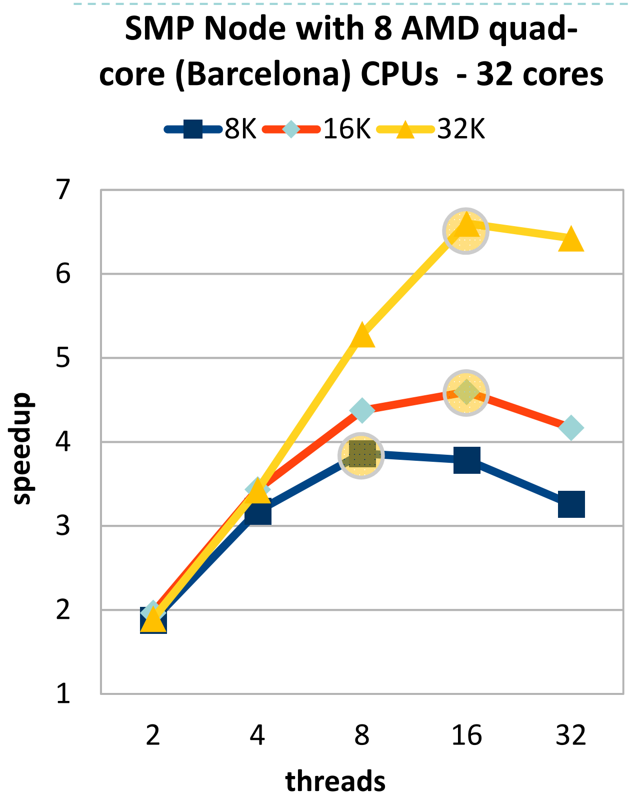
Modern processor and system architectures are so complex that it appears to be impossible “for a human being” to manually find best code transformation sequences for a program to optimize performance.

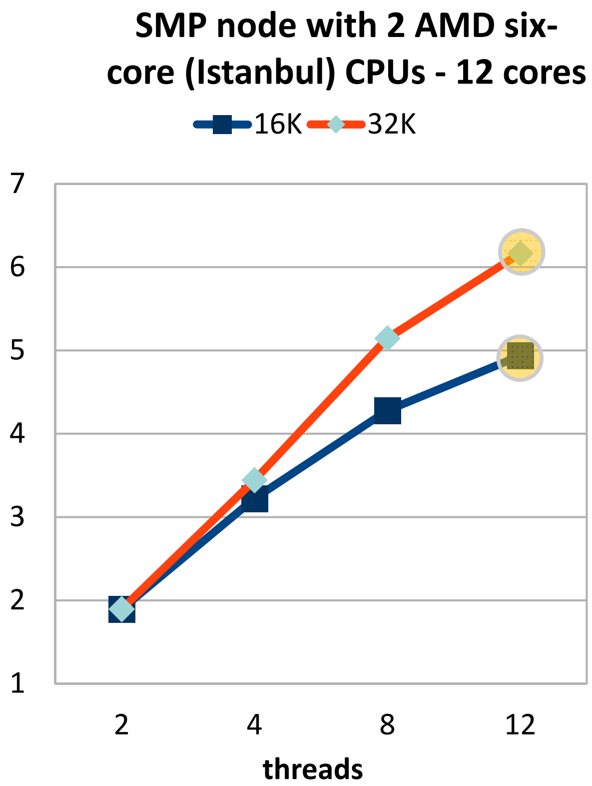
Impact of parallelism on tiling for matrix multiply relative execution time

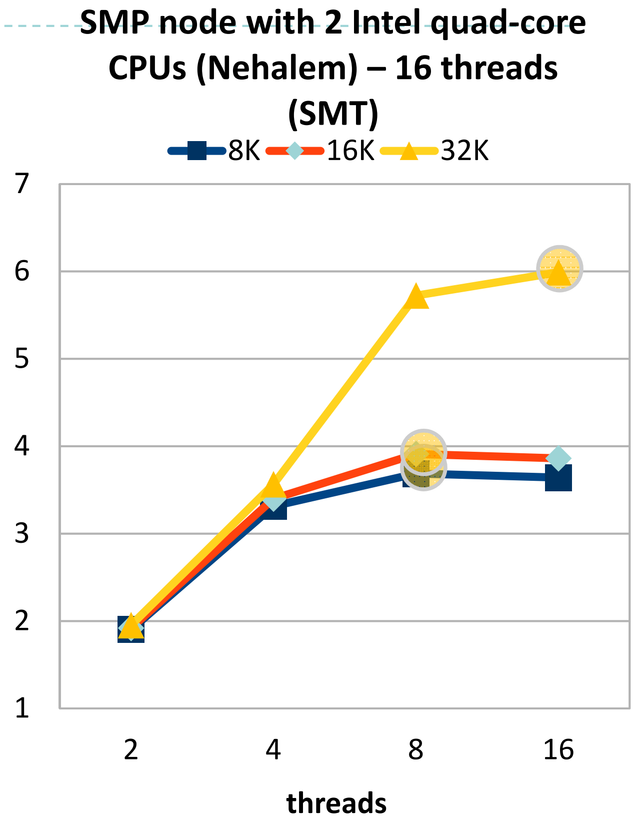
- MM-IJK loop order
  - 1400x1400
- Intel Westmere Arch.
  - 4 x 10 cores
  - 30MB L3 / Socket
- Brute Force Search
  - Tile Sizes I/J/K
  - # threads



ADI OpenMP Comparison







What is the optimal number of cores to use?

- Performance impact: CPU architecture, cache size and memory hierarchy
- Ideal number of threads requires knowledge about the program, architecture, and input data.

Search Space for Code Optimizations and Parallelization Strategies

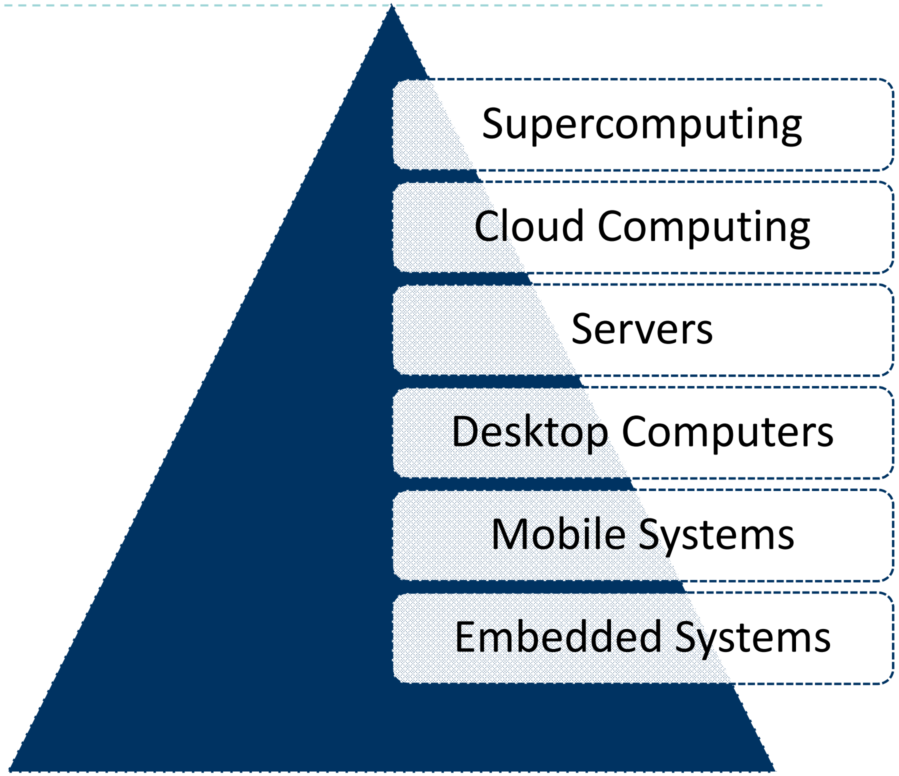
- code transformations
  - loop tiling, scheduling, data transpose, data distribution, …
- library settings
  - MPI has 100s of parameters
- target compiler flag setting
  - memory size, optimization flags, binary optimizations
- target machine configuration
  - nr. cores, frequency and power settings, turbo boost

Gigantic search space to explore for manual optimization.

➡ auto-tuning

Objectives for Optimization

- execution time
- memory footprint
- disc usage
- energy/power
- economic costs
- program size
- reliability
- security



- Auto-tuning for multiple objectives

Outline

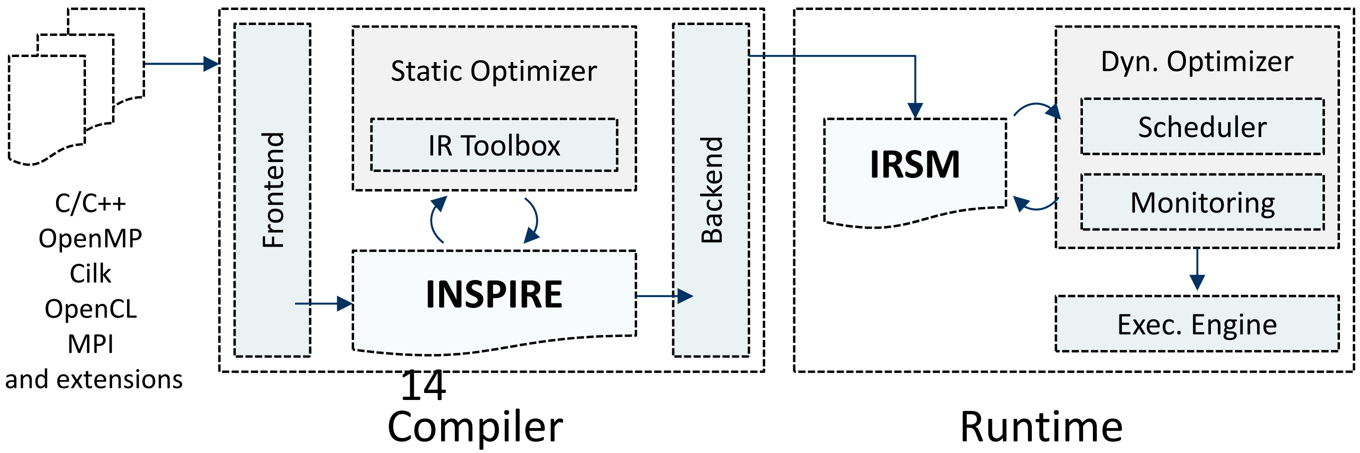
- Insieme Compiler Framework
- Multi-Objective Optimization and Auto-tuning
- Runtime / Efficiency / Energy
- Region-aware auto-tuning
- Experimental Results
- Conclusions

Insieme Compiler Framework

Insieme

Compiler and Runtime Research Platform

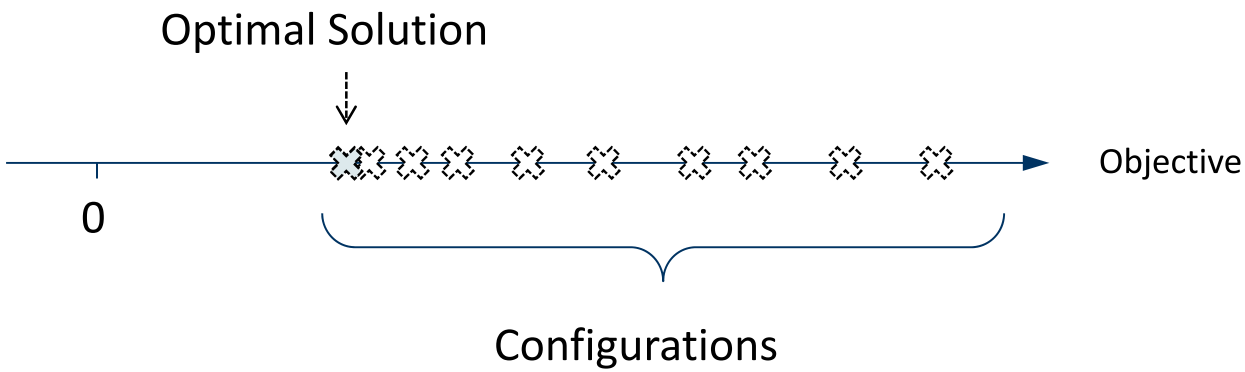
- *Insieme Compiler*
  - Source to Source Compiler for Parallel Codes
  - C/(C++) OpenMP, MPI, OpenCL, Cilk
  - Uniform Internal Representation (INSPIRE)
  - Analyses and Transformations Frameworks
    - e.g. Polyhedral Model or Pattern based
- *Insieme Runtime*
  - scheduling & runtime auto-tuning research
  - compiler-aided decision making processes
  - external load



Multi-Objective Optimization

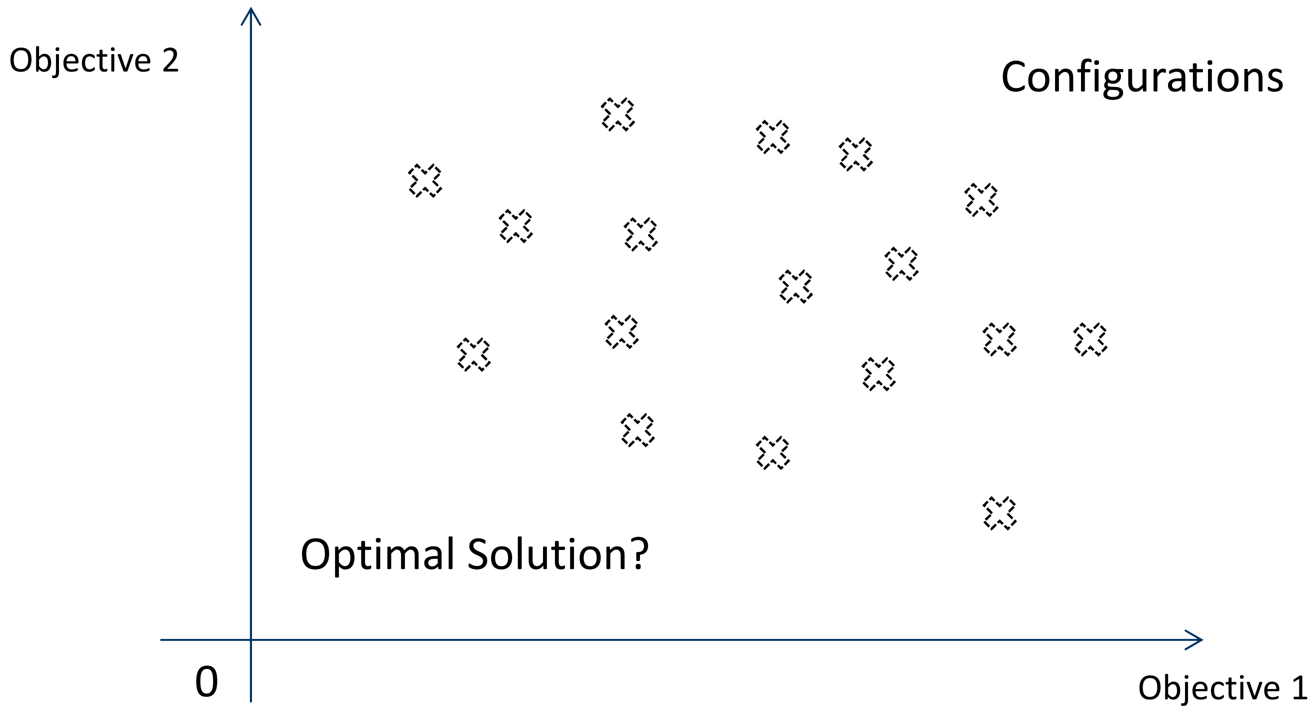
Mono-Objective Optimization

- configuration: specific code version with specific setting of tunable parameters



$$f : C \to \mathbb{R}$$

Multi-Objective Optimization

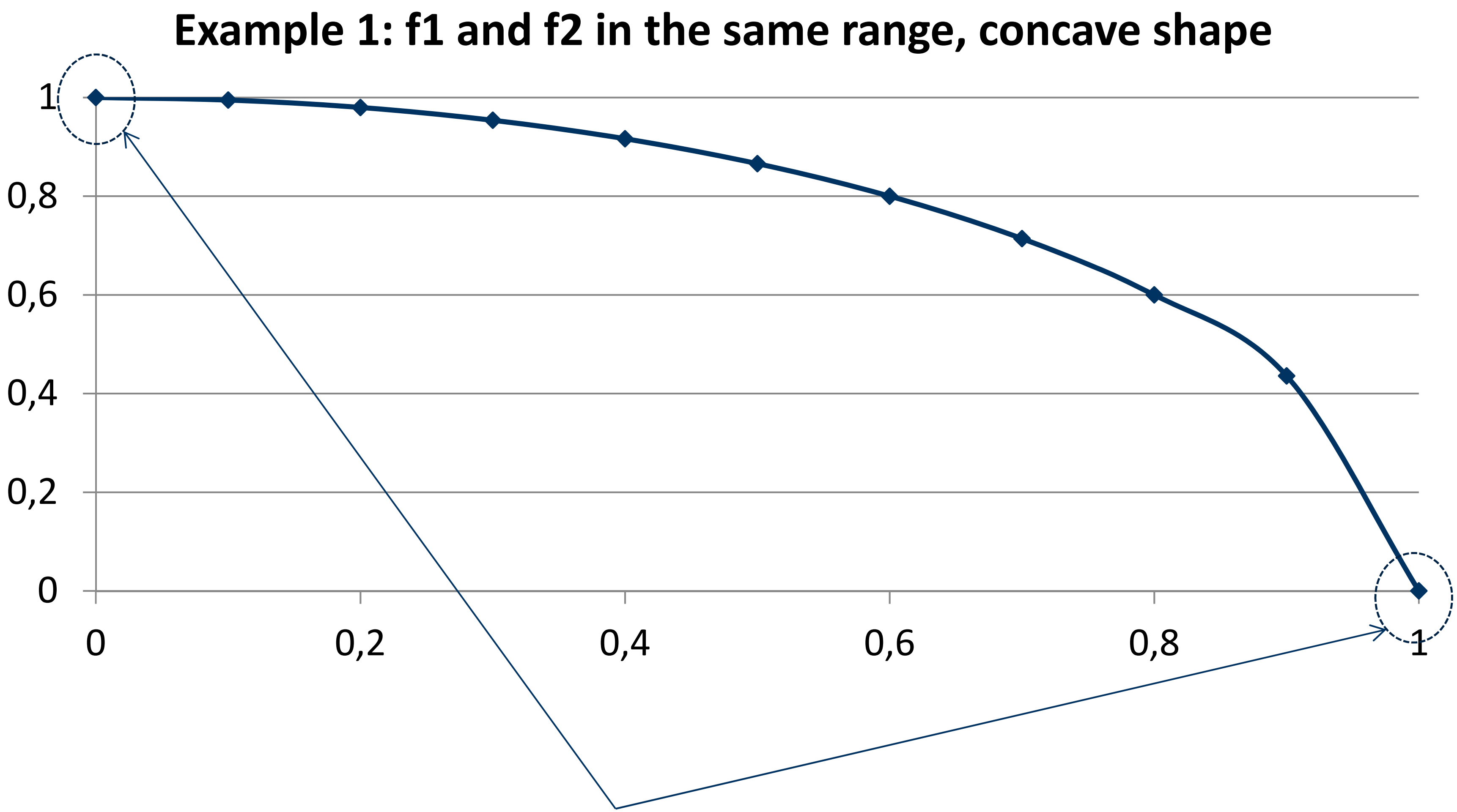


$$f: C \rightarrow \mathbb{R}^n$$

Multi-Objective Optimization with a single Function

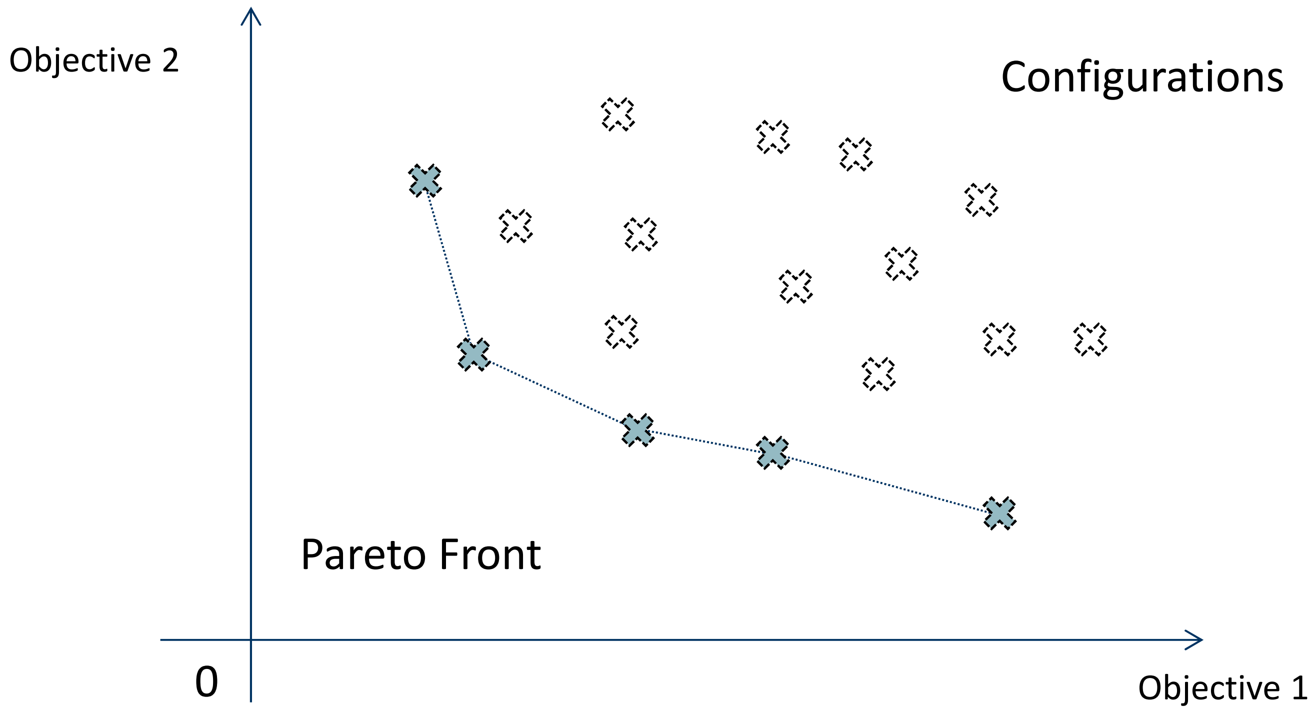
- Merge all objectives in a single function
  - Objectives are combined by means of weights (expressing user preferences)
  - Example:
    - Minimize energy consumption (f1) and runtime (f2) can be defined as minimizing the function
      - F = a * f1 + b * f2, where (a,b) are user preferences for that objective function
  - Drawbacks
    - F depends on the shape of the front and the value ranges of the objective functions
    - Therefore, there is no guarantee of obtaining the desired solution

Multi-Objective Optimization with a single Function



Two solutions are optimal for the preferences (0.5,0.5)
Which solution will be determined depends on the solver.

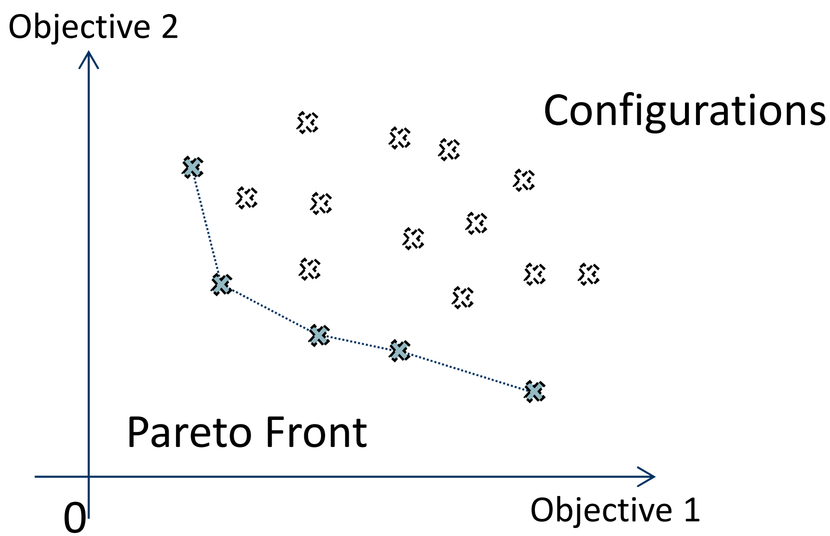
Multi-Objective Optimization with a Pareto Front



$$f: C \rightarrow \mathbb{R}^n$$

Multi-Objective Optimization with a Pareto Front

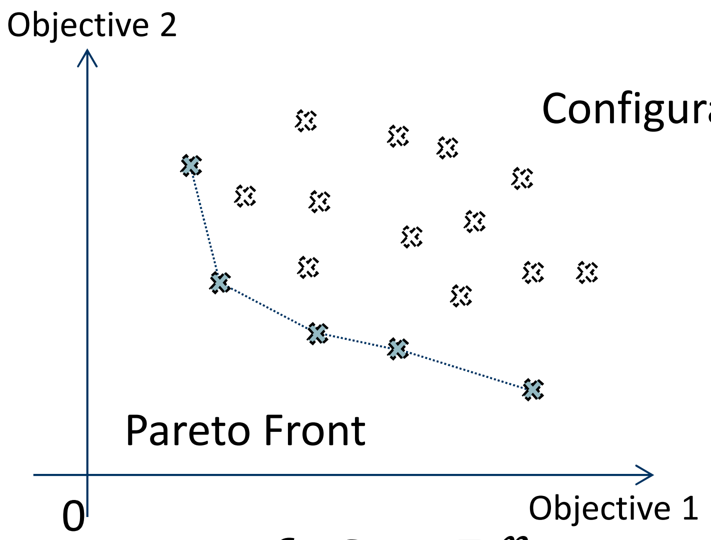
- Goals of Optimizer:
  - Push front toward ideal point
  - Evenly distributed points on front
  - Large number of points on front



$$f: C \to \mathbb{R}^n$$

Selection of a Solution

$$g(c) = \sum_{i=1}^{o} w_i \frac{f_i(c) - f_i^{\min}}{f_i^{\max} - f_i^{\min}}$$



$f : C \rightarrow \mathbb{R}^n$

Optimization Objectives

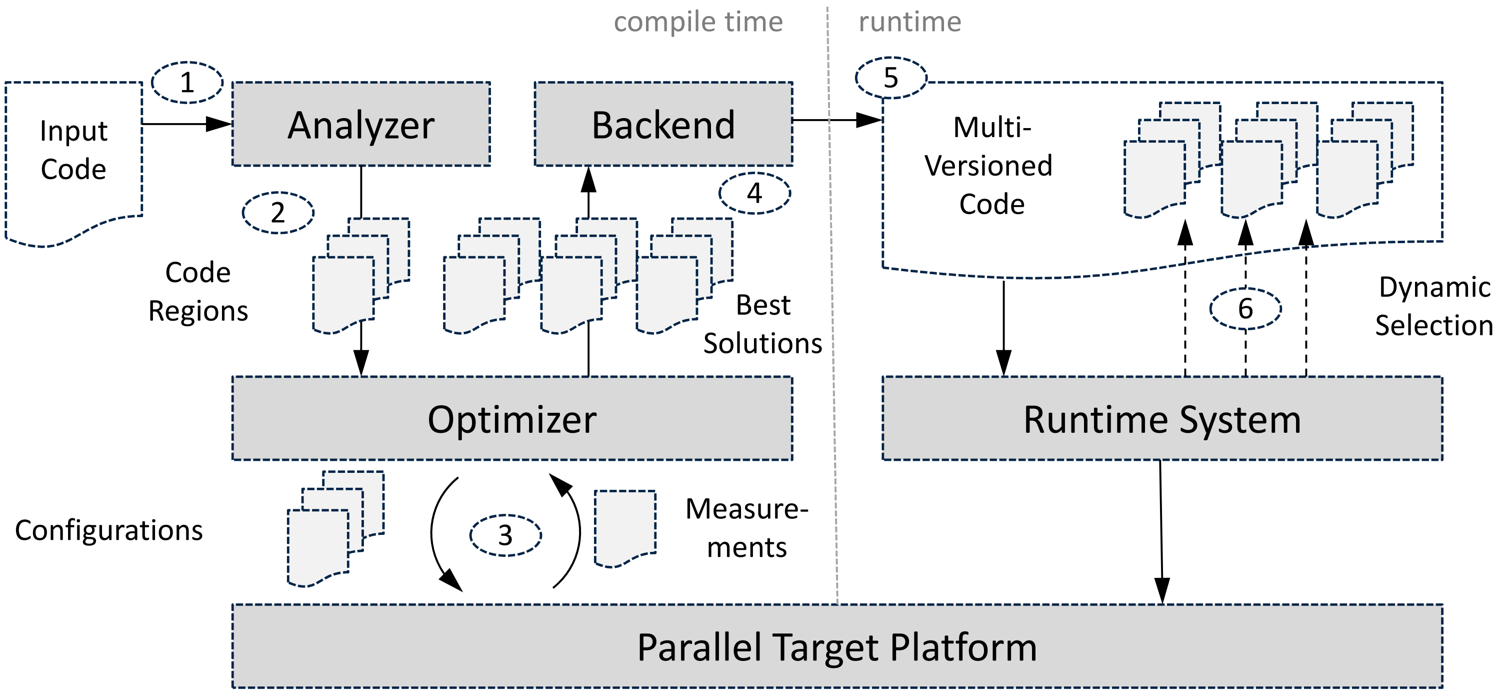
- Execution time
- Resource efficiency
- Energy consumption
- Memory footprint
- Economic costs
- …

Insieme can simultaneously handle an arbitrary number of measurable objectives.

Compiler Framework

Laying the Foundation

Insieme Multi-Objective Auto-Tuning Compiler Architecture



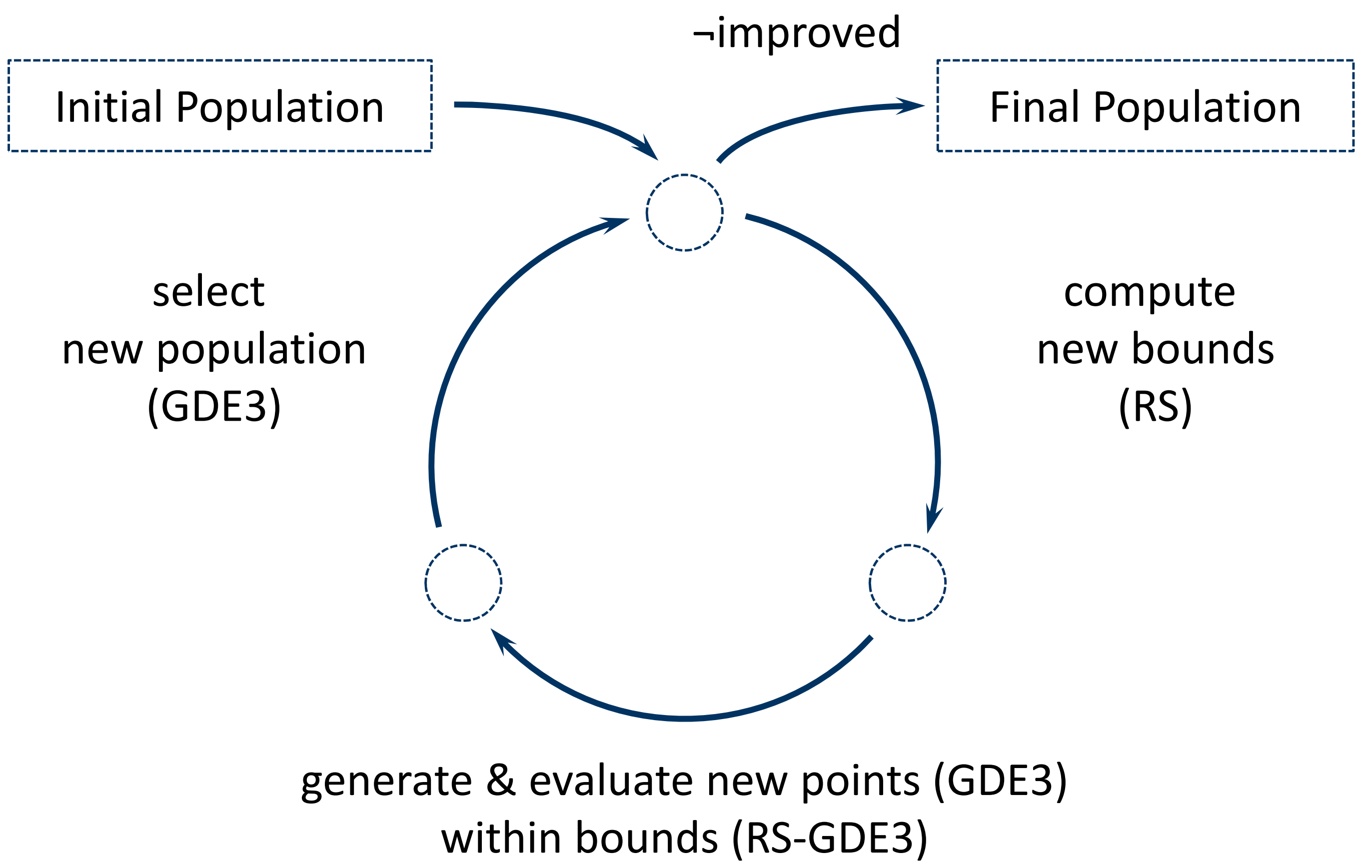
Optimization Algorithm

- Our Algorithm: **RS-GDE3**

- Combination of:
  - Metaheuristics: Evolutionary Computation:
    Differential Evolution algorithm *GDE3*
    - ***Generalized Differential Evolution***
    - => conducts the actual **iterative search**

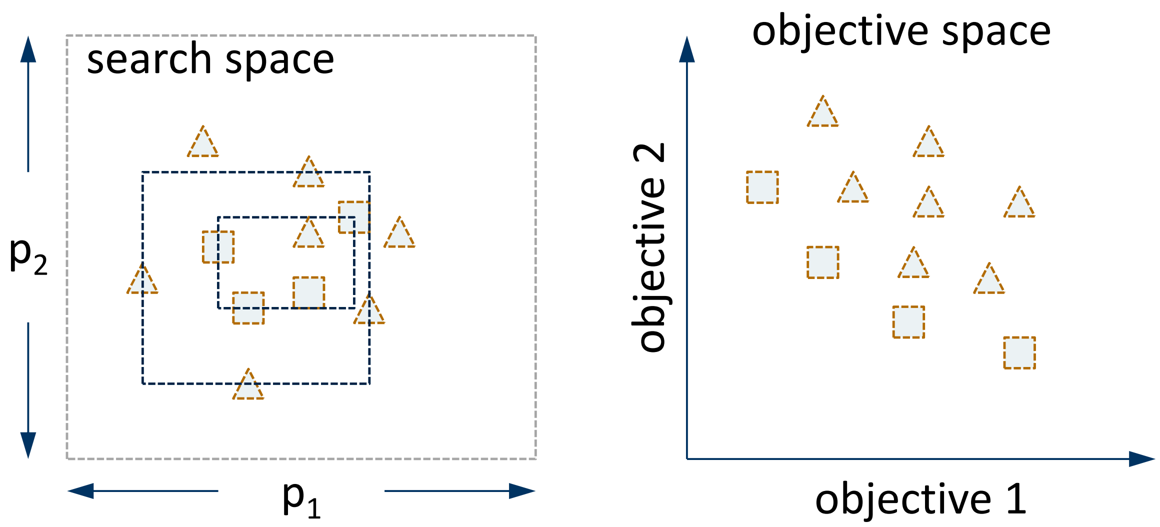
  - ***Rough Set*** based **Search Space Reduction**
    - => domain independent

Optimization Algorithm (2)

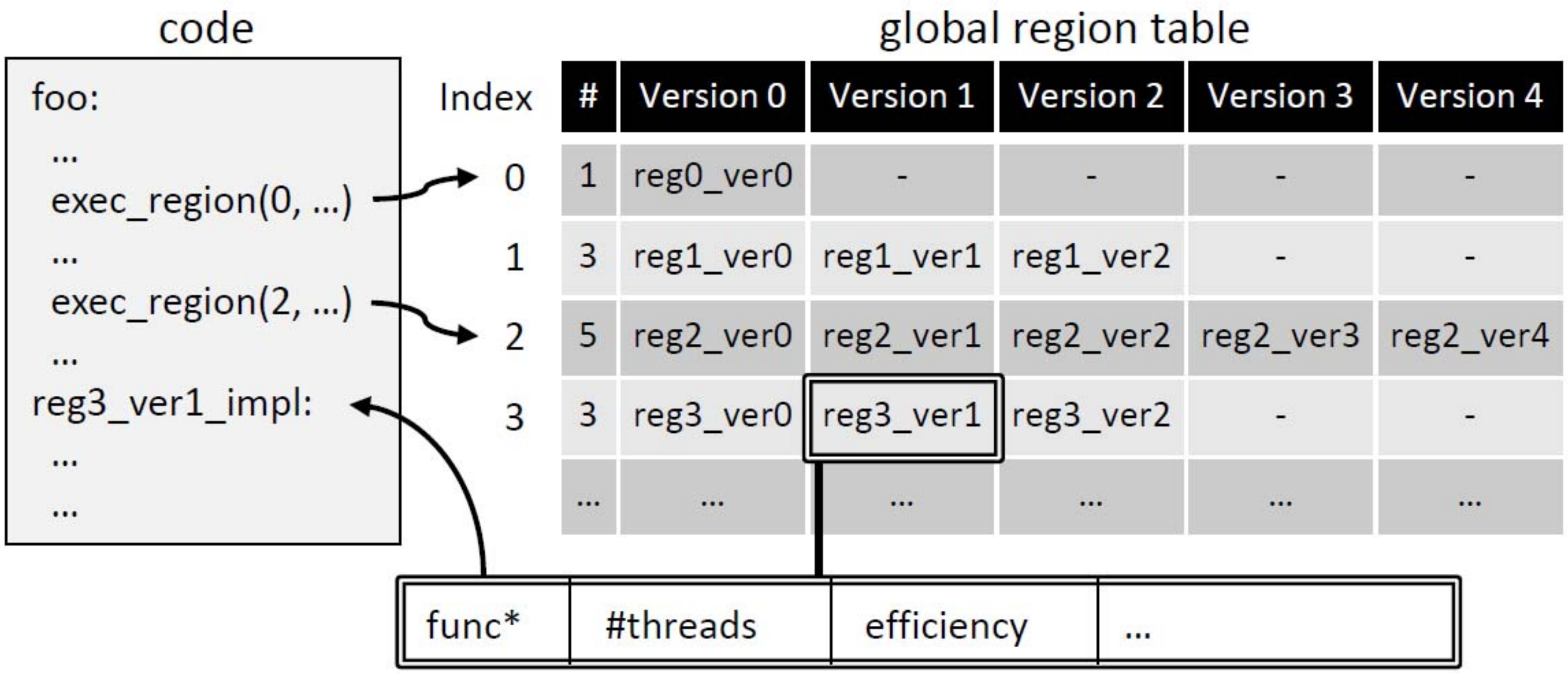


Search Space Reduction (RS)

- Compute bounds by ..
  - Obtaining bounding box of current front
  - Extend to enclosing dominated points



Runtime Version Selection



Experimental Results

Energy as an Optimization Objective

- Optimize 3 objectives simultaneously
  - Energy
  - Runtime
  - Efficiency (costs $c(x) \rightarrow x*t_p(x)$)

- Extra tuning parameter
  - 2/3D Tiling, #threads, **Clock Frequency**
  - Tradeoff between high (fast) energy-demanding frequencies and low (slow) frequencies
  - Larger search space than in the previous experiment

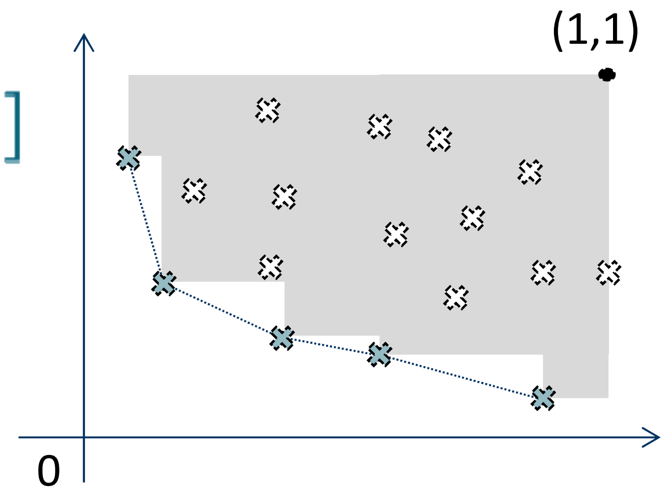
- Clock frequency modified through DVFS
  - Dynamic Voltage and Frequency Scaling

Experimental Setup

- Target architecture:
  - 4 x Intel Xeon SandyBridge with 8cores each / 8x20MB L3 Cache
  - 128 GB of RAM
  - Available clock frequencies from 1.2GHz to 2.6GHz
- Tuning Algorithms:
  - Hierarchical search– search along regular grid
  - RS-GDE3 (our algorithm)
  - Random Search
- Code
  - Matrix multiplication
  - N-body simulation

Comparing Algorithms

- Quality of Solution $\boldsymbol{S}$:
  - Hypervolume: $\boldsymbol{V}(\boldsymbol{S}) \in [0 \dots 1]$
  - Number of Points: $|\boldsymbol{S}|$

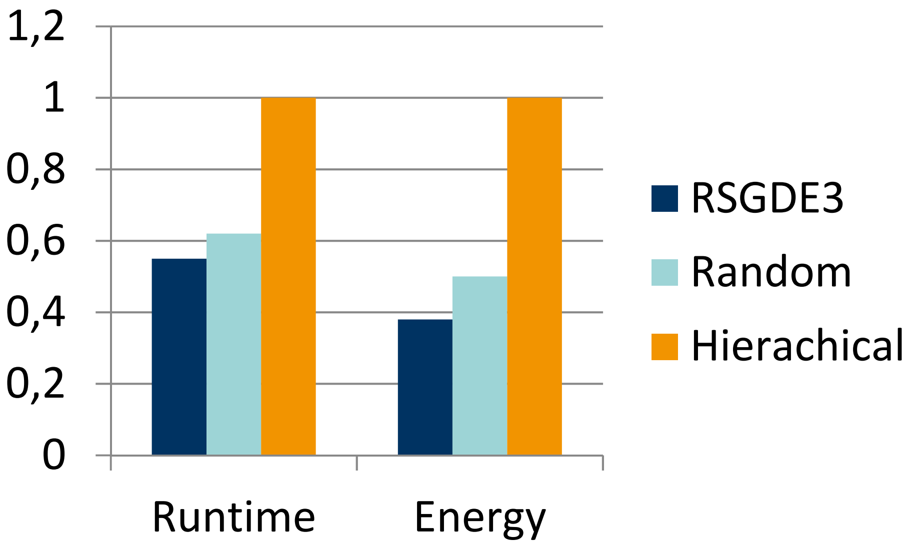


- Efficiency of Algorithm:
  - Number of evaluated Configurations $\boldsymbol{E}$

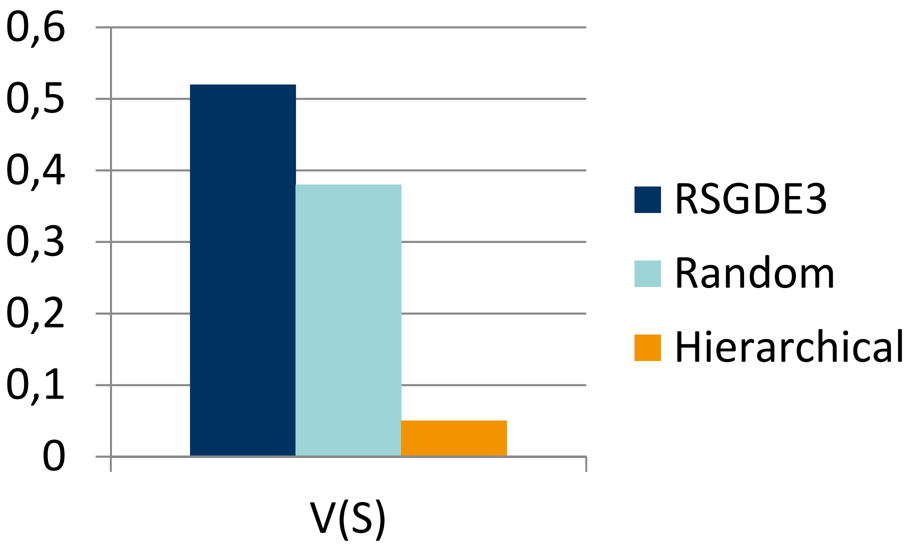
- For stochastic Algorithms: $\overline{\boldsymbol{V}(\boldsymbol{S})}$, $\overline{|\boldsymbol{S}|}$ and $\overline{\boldsymbol{E}}$

Obtained Results

- RSGDE3 only requires to evaluate less than 0.000013% of all possible configurations
- Comparing the fastest solutions
  - > 70% more energy efficient than hierarchical
    - 45 % faster
  - 24 % less energy consumption than random search
    - 14 % faster
- The hypervolume also shows the higher quality of the Pareto fronts computed by RSGDE3

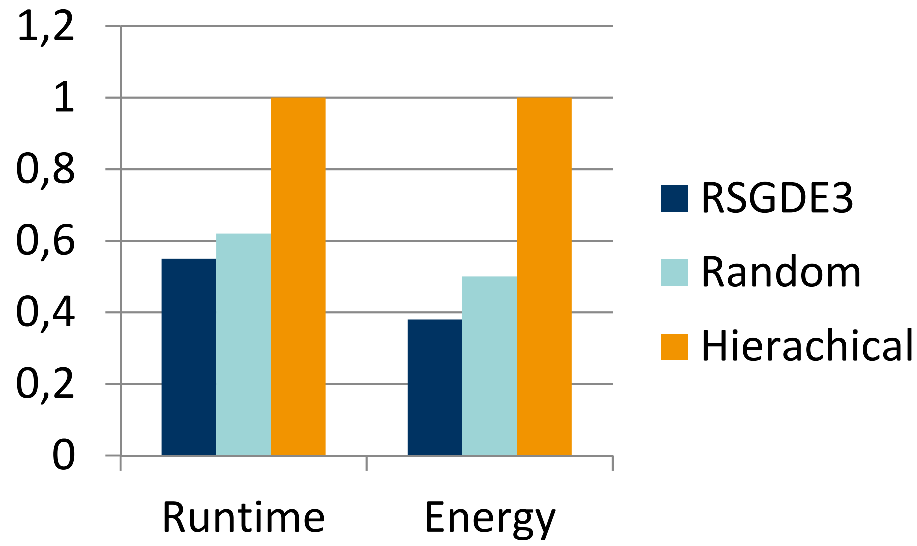


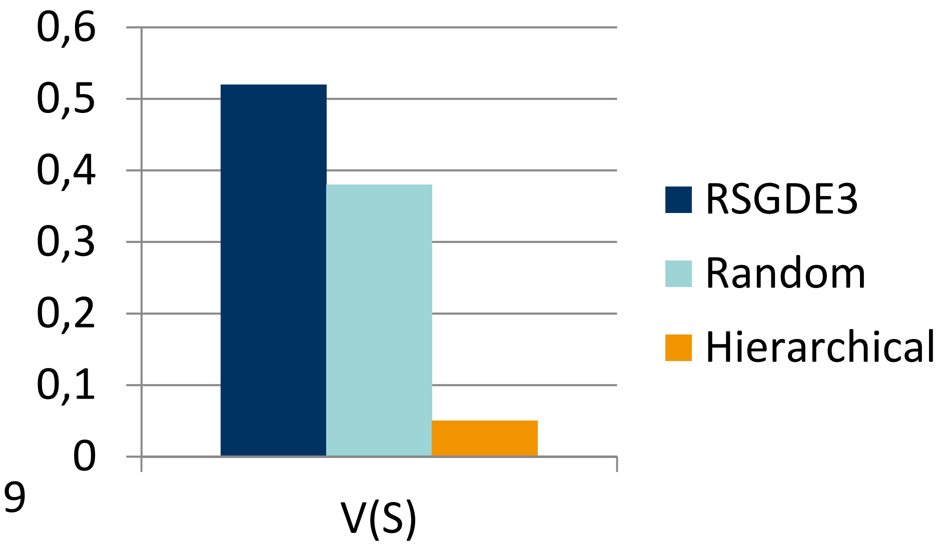
Data normalized to the (0 ,1) interval



Obtained Results (2)

- RSGDE3 only requires 6% of the evaluations performed by Random and hierarchical
- Random reports better results than hierarchical
  - Showing the non suitability of hierarchical search for exploring extremely large search spaces



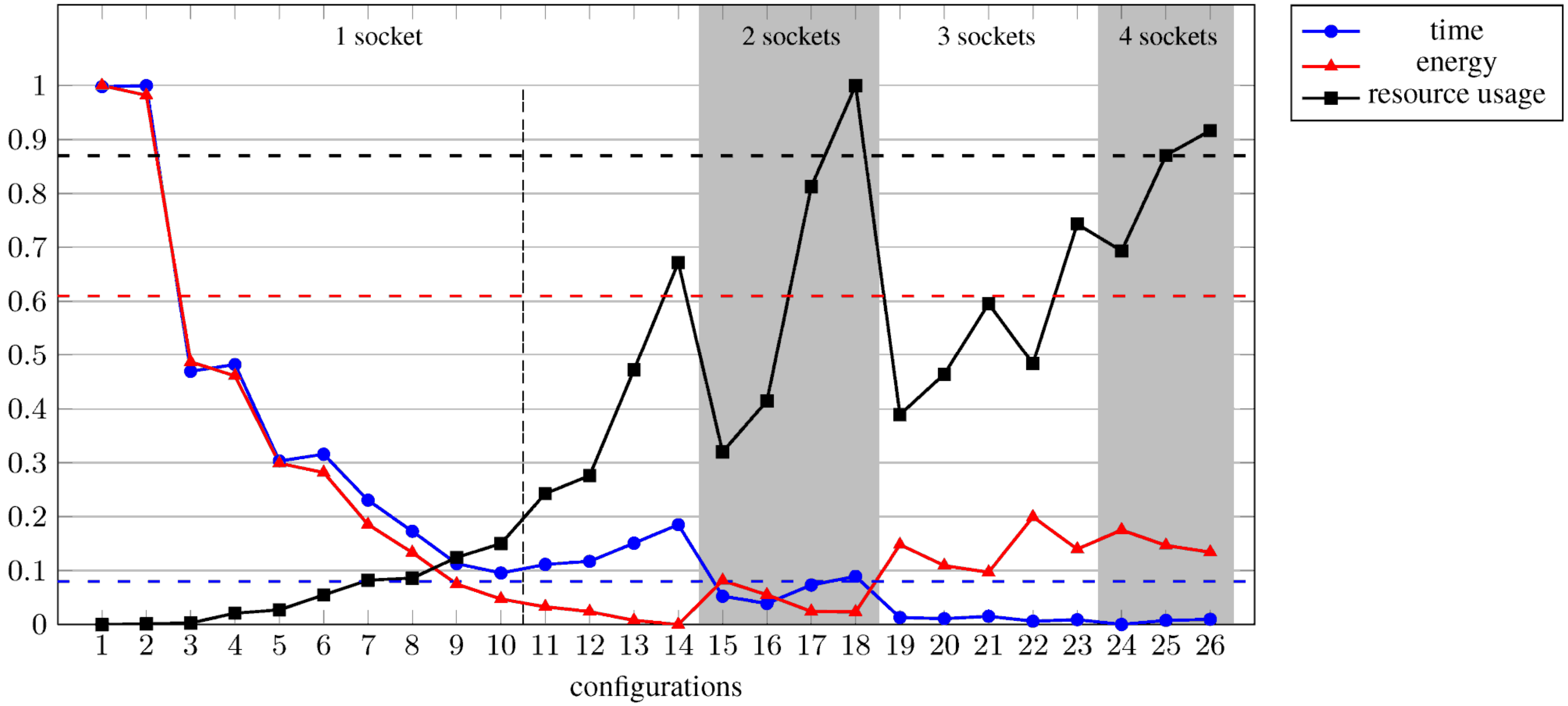


Performance Comparison

Benchmark	Hierarchical Search				Random Search			
	N	|S|	|S|'	V(S)	N	|S|	|S|'	V(S)
mm	18432	18	2%	0.00	15000	4.4	0%	0.33
dsyrk	18432	21	5%	0.00	15000	2.2	11%	0.17
jacobi-2d	15876	31	78%	0.69	15000	17.2	5%	0.55
3d-stencil	15876	30	22%	0.75	15000	24.8	60%	0.61
n-body	15876	26	0%	0.50	15000	30	17%	0.70

Benchmark	RS-GDE3			
	N	|S|	|S|'	V(S)
mm	956.2	23.4	98%	0.48
dsyrk	1149.6	24.8	98%	0.32
jacobi-2d	1243.6	29.8	75%	0.76
3d-stencil	981.4	28.2	77%	0.76
n-body	1801.4	29.6	87%	0.77

Trade-offs among 3 Objectives for n-body



Region-Aware Auto-Tuning

- Most existing auto-tuners
  - global tuning finds fixed parameter values for the entire program.
  - assumes that fixed parameter values achieve best performance across all regions.
- Programs may consist of many code regions
  - code region specific parameter values may yield best performance
  - changing parameter values implies overhead
- Region based Auto-tuning increases search space

Motivating Example

```
#pragma omp parallel for
for (int i=0; i<N; i++) {
   for (int j=0; j<K; j++) {
       for (int k=0; k<M; k++) {
          C[i][j] += A[i][k] * B[k][j];
       }
   }
}
```

scales well

```
for (int i=0; i<N/4; i++) {
   for (int k=0; k<M/4; k++) {
      #pragma omp parallel for
      for (int j=0; j<K/4; j++) {
         C[i][j] += A[i][k] * B[k][j];
      }
   }
}
```

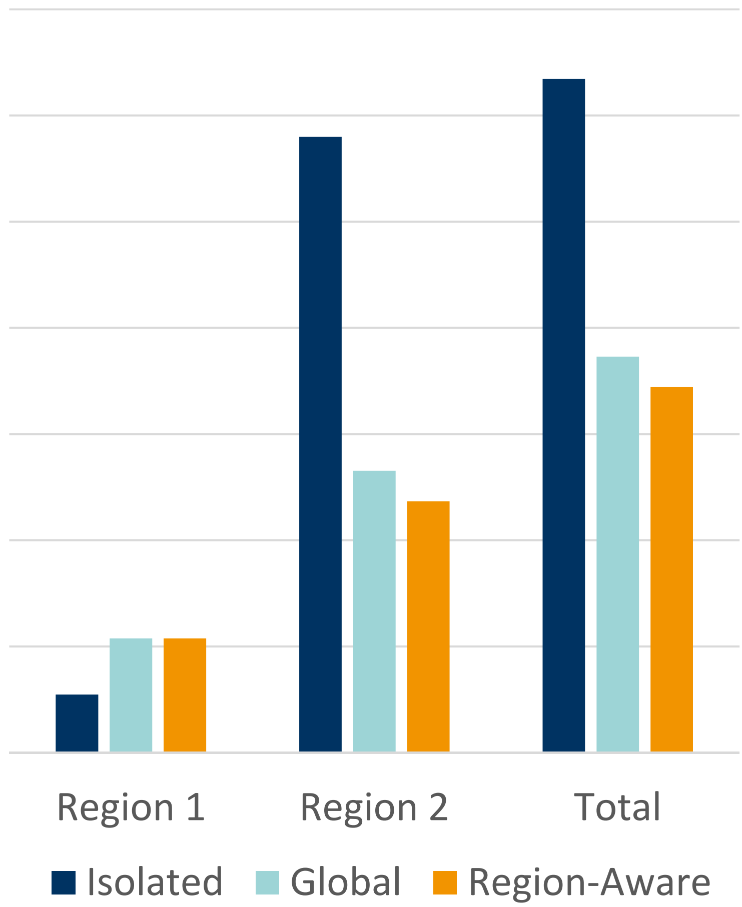
doesn,t scale well

Region-Aware Auto-Tuning Approaches

- Isolated
  - Tune regions individually following the control flow
  - Ignores overhead for changing parameter values
- Global
  - Same parameter values for the entire program
- Region-Aware
  - tunes all regions simultaneously
  - Individual parameter values for each region
  - Optimizes the overal program performance
  - Considers overhead for changing parameter values

Motivating Example

	Isolated	Global	Region-Aware
#Threads Region 1	20	10	10
#Threads Region 2	2	10	7
Region 1	546	1075	1075
Region 2	5798	2652	2366
Total	6344	3727	3442



Challenges of Region-Aware Auto-Tuning

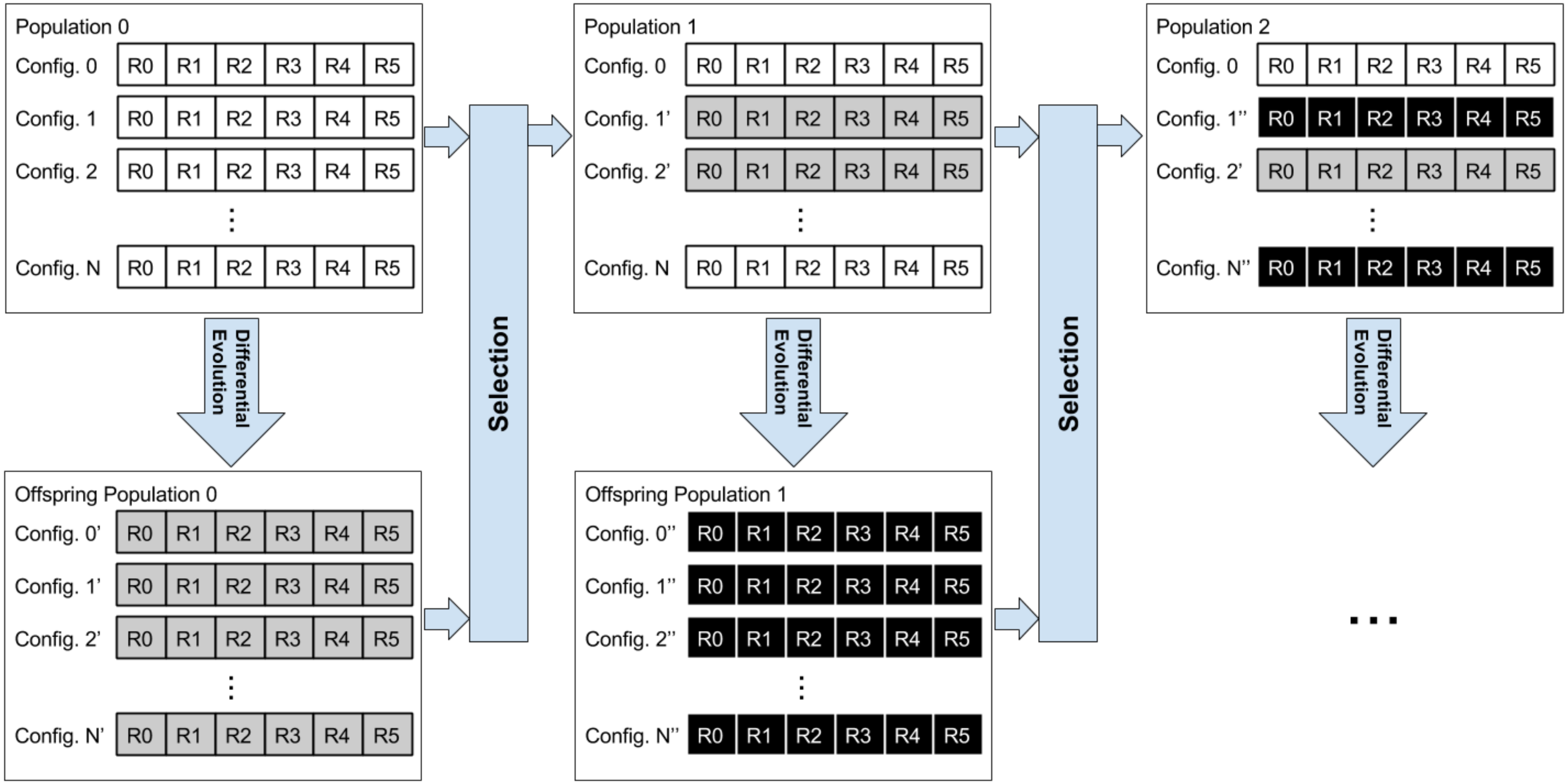
- Region dependencies
  - Settings of one region may affect performance of others
- Consistent goal
  - Settings of all regions should optimize for the same trade-off solution
- Huge search space
  - Growing exponentially with number of regions

Extentions to the Region-Aware RS-GDE3*

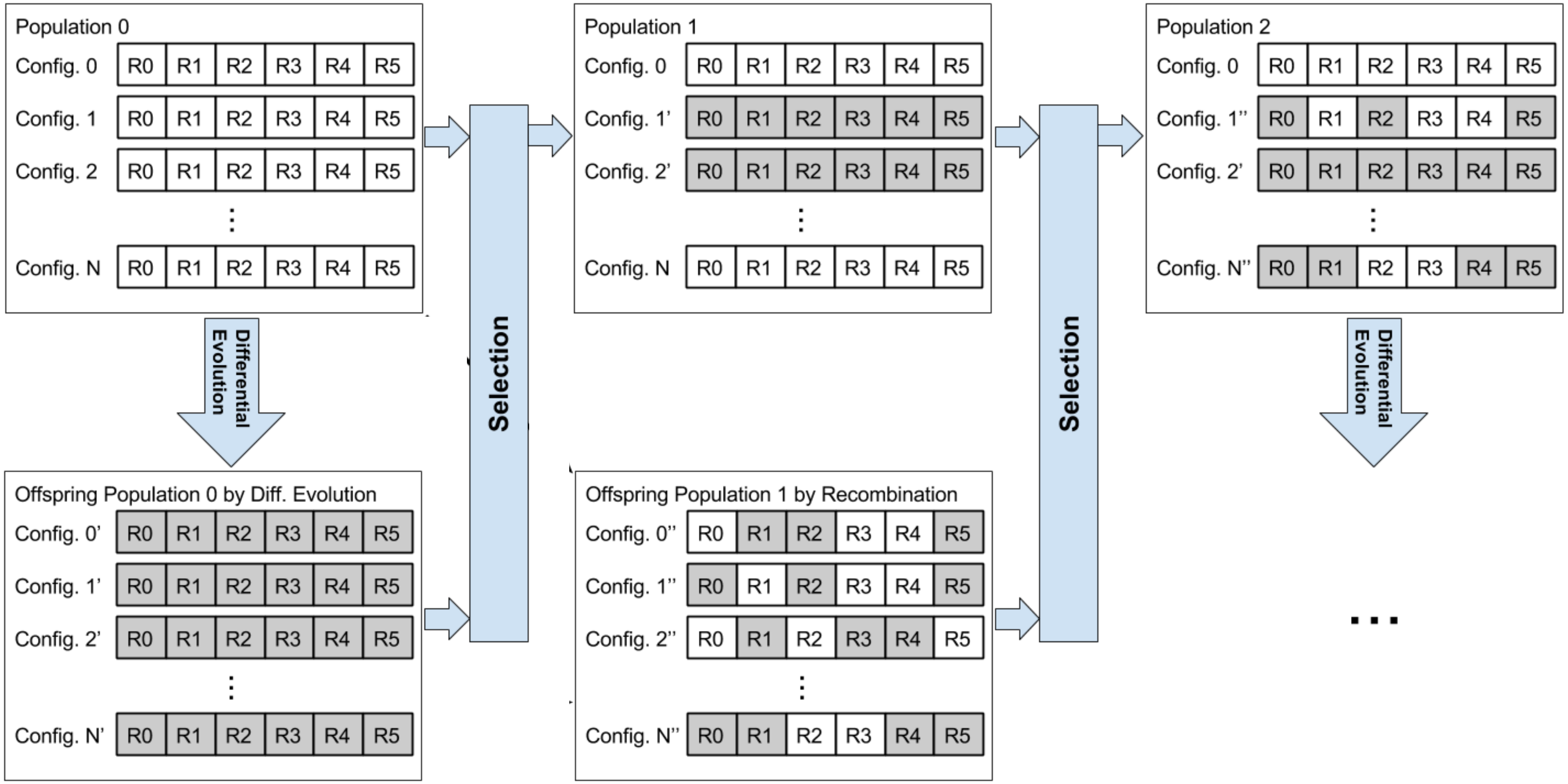
- Global pre-tuning phase
  - Using global tuner for first iterations
  - Reduces initial search space
  - Quickly converge to a reasonable starting point
  - Avoid overheads between regions
- Recombination
  - Avoid purging of good local solutions
  - Combine parameter values optimizing same objective
  - Maximizing the likelyhood of reusing good parameter values per regions

* K. Kofler, J. Durillo, P. Gschwandtner, T. Fahringer. A region-aware multi-objective auto-tuner for parallel programs. P2S“, Bristol, UK, Aug. 2017.

Region aware GDE3



Region aware GDE3 with Recombination



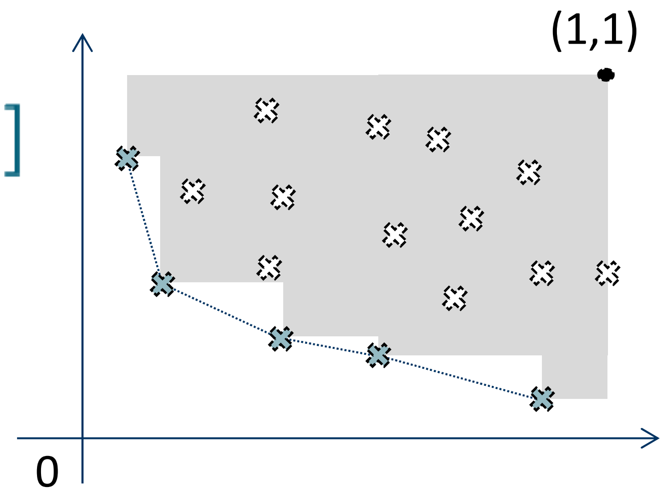
Experimental Results

Architectures

- Two multi-socket Intel Xeon
  - Intel Xeon E5-2690 v2 (Ivy Bridge-EP), 20 cores
  - Intel Xeon E5-4650 CPU (Sandy Bridge-EP), 32 cores
- Three benchmarks
  - bt (block tri-diagonal solver from NAS parallel benchmarks)
  - mg (three-dimensional discrete Poisson equation from NAS parallel benchmarks)
  - Heated-plate (stencil-code solving the steady heat equation)

Comparing Tuners

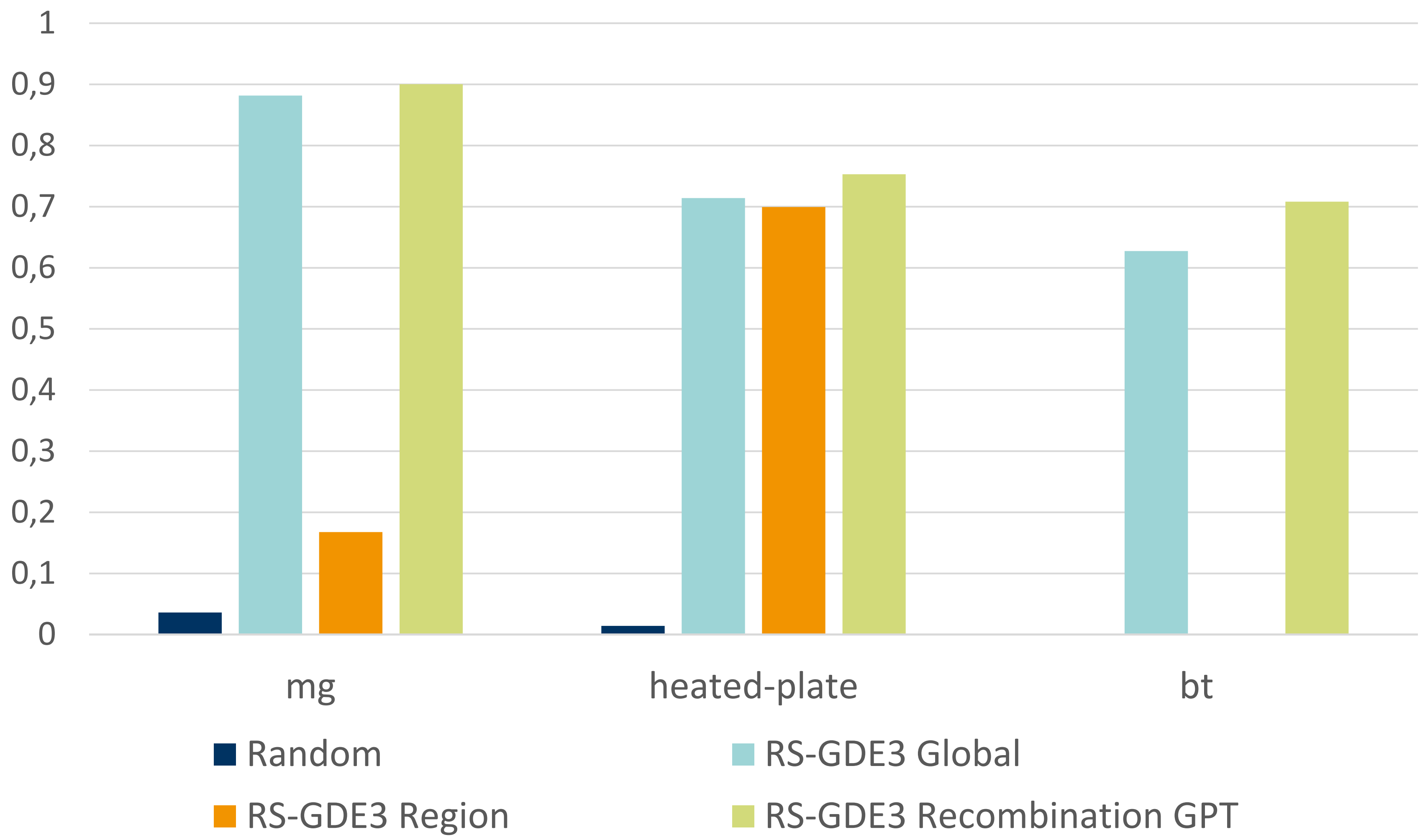
- Quality of Solution $\boldsymbol{S}$:
  - Hypervolume: $\boldsymbol{V}(\boldsymbol{S}) \in [0 \dots 1]$
  - Number of Points: $|\boldsymbol{S}|$



- Efficiency of Algorithm:
  - Number of evaluated Configurations $\boldsymbol{E}$

- For stochastic Algorithms: $\overline{\boldsymbol{V}(\boldsymbol{S})}$, $\overline{|\boldsymbol{S}|}$ and $\overline{\boldsymbol{E}}$

Hypervolume

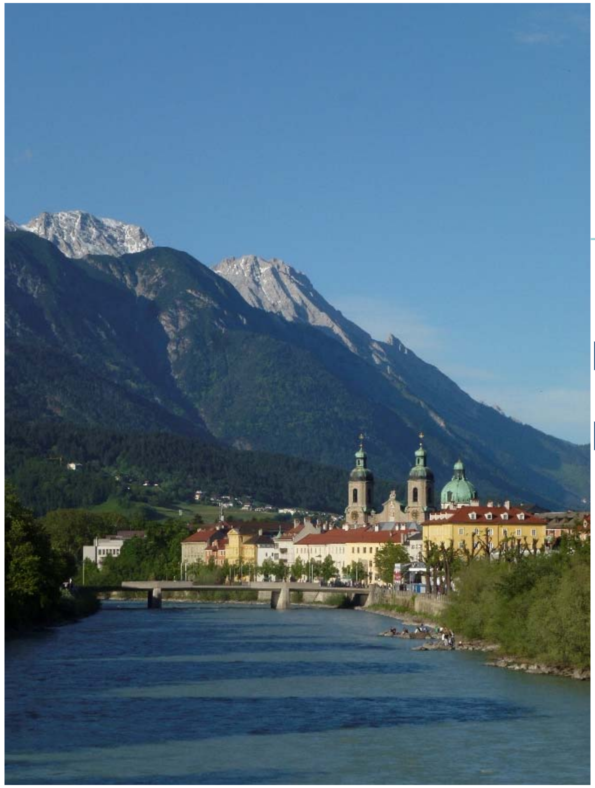


Tuning Time

	mg	heated-plate	bt
Random	26913.5	30706	31431.5
RS-GDE3 Global	17871	12365	21246
RS-GDE3 Region	25088	16712.5	30997.5
RS-GDE3 Recombination GPT	21209	13237	25162.5

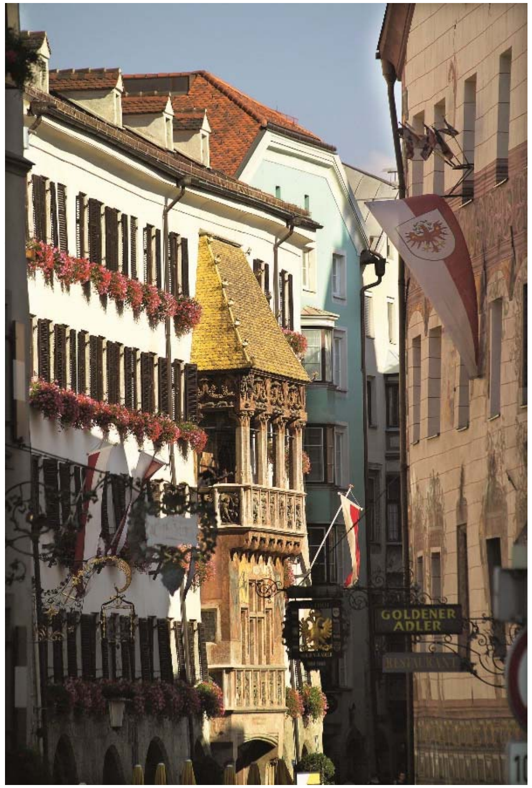
Conclusion

- motivated the **usefulness for multi-objective** optimization and auto-tuning within compilers
- developed a suitable **generic optimizer algorithm** RS-GDE3 for an arbitrary number of objectives
- demonstrated its **effectiveness** on tuning tiled parallel loops for 3 objectives:
  - runtime
  - resource utilization
  - Energy
- Region aware auto-tuner
  - Up to 7x faster than un-optimized code
  - Up to 17% improvement over traditional auto-tuner



Thank you!

- Further Information:
- http://www.insieme-compiler.org/



- Insieme Funding as part of European projects:
  - H2020 FETHPC AllScale
  - Chist-era Gemsclaim
  - Interreg IV En-act
  - NoE Hipeac
  - ICT Cost Action NESUS