Optimization of LLVM-Based Code using Multi-Objective Evolutionary Algorithms

Bernabé Dorronsoro
University of Cadiz

Sebastien Varrette
University of Luxembourg

Outline

- Context and motivation
- The code optimization problem
- Introduction to multi-objective optimization
- The Evo-LLVM compiler framework
- Preliminary results
- Conclusion and perspectives

Outline

- Context and motivation
- The code optimization problem
- Introduction to multi-objective optimization
- The Evo-LLVM compiler framework
- Preliminary results
- Conclusion and perspectives

Energy in Today's Computing Systems

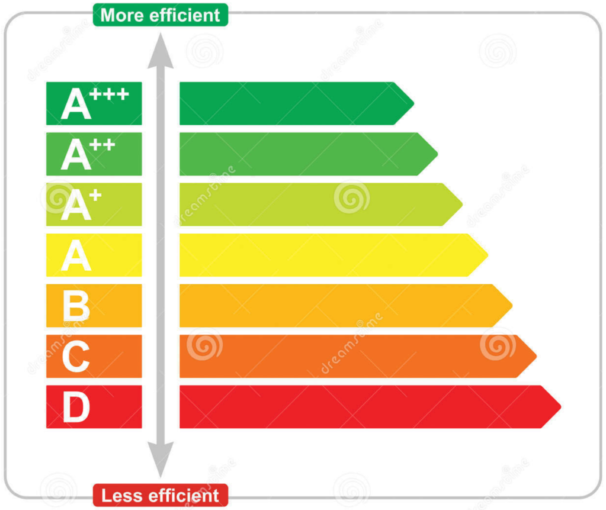
- Energy consumption
  - Key issue in modern computer systems
  - Increasing computing / storage needs
    - Virtualization, simulation, Big Data analytics, …
- Energy efficiency challenge
  - 2020 Exa-scale challenge: 1 EFLOPS in 20 MW
    - Today's most efficient supercomputer: 314 MW
  - Foreseen combined solution
    - Involving HW / Middleware / Software improvements

Energy in Today's Computing Systems

- Achieving energy efficiency in HPC
  - Reduce operating costs
  - Reduce impact on environment
  - Become more competitive

Energy in Today's Computing Systems

- Not only HPC and large servers are affected
  - Personal computers
  - Battery powered devices
  - Any other electronic devices
    - Internet of things
- Advantages
  - Longer operation times
  - Adding sensors and computing capacity to things
    - Making intelligent things



Energy Management

- Recent HW supports energy management at various levels
  - Dynamic scaling of the power (or freq) of CPU/ Memory
  - Integrated way to handle idle state
  - Embedded sensor to measure energy and performance metrics
- Power drainage of a system is closely related to workload

Energy Management

- Reserach question

Can we produce energy aware workload through source code evolution?

Energy Management

- In this talk: EvoLLVM
  - Goal: Evolve a given source code to produce energy-aware versions
  - Tools
    - LLVM Compiler Infrastructure
    - Multi-objective optimization algorithms
  - Features
    - Combining energy and performance metrics for evaluation of programs
    - Software is optimized for a specific architecture

Outline

- Context and motivation
- The code optimization problem
- Introduction to multi-objective optimization
- The Evo-LLVM compiler framework
- Preliminary results
- Conclusion and perspectives

Code optimization

- Implemented using sequence of optimizing transforms
  - Produce a *semantically equivalent* output program
  - Transforms order matters
  - **NP-complete problem***
- Thus modern compilers (GCC, LLVM) rely on static heuristics
  - Involves subset of transformations producing good results **in general**

* A. Nisbet. GAPS: A Compiler Framework for Genetic Algorithm (GA) Optimised Parallelisation. In HPCN Europe, pages 987–989, 1998

Code Transformation Examples

- Loop unrolling of rate K

<table>
<tr><th>Normal loop</th><th>After loop unrolling</th></tr>
<tr><td>

```
int x;
for (x = 0; x < 100; x++)
{
    delete(x);
}
```

</td><td>

```
int x;
for (x = 0; x < 100; x += 5)
{
    delete(x);
    delete(x + 1);
    delete(x + 2);
    delete(x + 3);
    delete(x + 4);
}
```

</td></tr>
</table>

Code Transformation Examples

- Localize declaration

```
#include <stdio.h>

int main(){
    int i,j;
    int a [15][15];

    for(i=0;i<15;i++){
        for(j=0;j<15;j++){
            a[i][j] = i+j;
        }
    }

    for(i=0;i<15;i++){
        for(j=0;j<15;j++){
            a[i][j] = i+j;
        }
    }

    return 0;
}
```

(a) original program

```
int main(){
    int i;
    int a [15][15];

    for(i = 0; i <= 14; i += 1) {
        //PIPS generated variable
        int j;
        for(j = 0; j <= 14; j += 1)
            a[i][j] = i+j;
    }
    for(i = 0; i <= 14; i += 1) {
        //PIPS generated variable
        int j;
        for(j = 0; j <= 14; j += 1)
            a[i][j] = i+j;
    }
    return 0;
}
```

(b) transformed program

Code Transformation Examples

- Code flattening

```
#include <stdio.h>

int main(){
    int i;
    int a[4];

    for(i=0;i<4;i++){
        int k = i+5;
        a[i] = 5;
    }

    if (a[0] == 7){
        int k = a[1];
    }
    return 0;
}
```

(a) original program

```
#include <stdio.h>

int main() {
    int i;
    int a[4];
    //PIPS generated variable
    int k, k_0;
    k = 0+5;
    a[0] = 5;
    k = 1+5;
    a[1] = 5;
    k = 2+5;
    a[2] = 5;
    k = 3+5;
    a[3] = 5;
    if (a[0]==7)
        k_0 = a[1];
    return 0;
}
```

(b) transformed program

Code Transformation Examples

- Parallel loop generator

```
int foo(int a [15][15], b [15][15]){
    int i,j;
    int c [30];

    for (i=1;i<14;i++){
        for (j=1;j<14;j++){
            c[i+j]=a[i−1][j]+b[i][j]∗a[i][j+
        }
    }

    return 0;
}
```

(a) original program

```
int foo(int a [15][15],  int b [15][15]){
   int i, j;
   int c [30];
   for(i = 1; i <= 13; i += 1)
#pragma omp parallel for
   for(j = 1; j <= 13; j += 1)
       c[i+j] = a[i−1][j]+b[i][j]∗a[i][j+1];
   return 0;
}
```

(b) transformed program

About LLVM

- Collection of modular/reusable compiler and toolchain technologies
- Multiple LLVM front-ends. Ex: Clang
- Supports just-in-time optimization and compilation
- LLVM core
  - Intermediate representation (IR) of the program
  - 54 built-in transformations (called *passes*)

LLVM IR

```
int mul_add(int x, int y, int z) {
  return x * y + z;
}
```

$$\Downarrow$$

```
define i32 @mul_add(i32 %x, i32 %y, i32 %z) {
entry:
  %tmp = mul i32 %x, %y
  %tmp2 = add i32 %tmp, %z
  ret i32 %tmp2
}
```

Outline

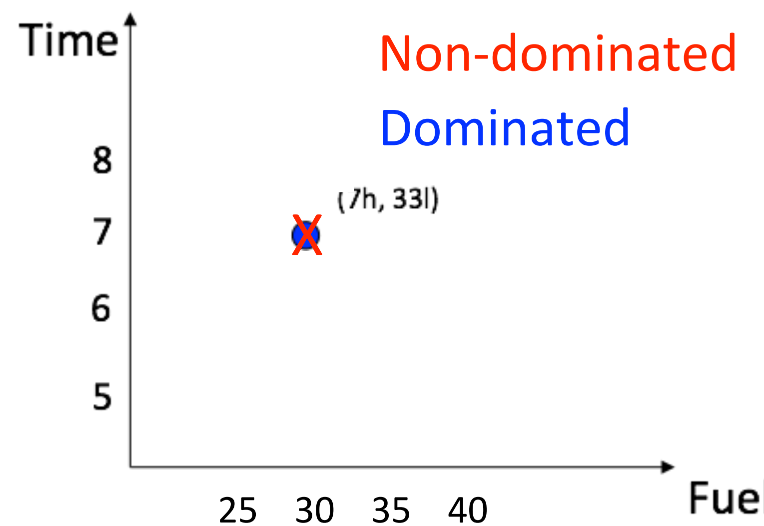
- Context and motivation
- The code optimization problem
- Introduction to multi-objective optimization
- The Evo-LLVM compiler framework
- Preliminary results
- Conclusion and perspectives

What is multi-objective optimization?

- Many real-world optimization problems require to optimize **more than one objective** at the same time
  - These objectives are usually in conflict among them
  - Improving one means worsening the others

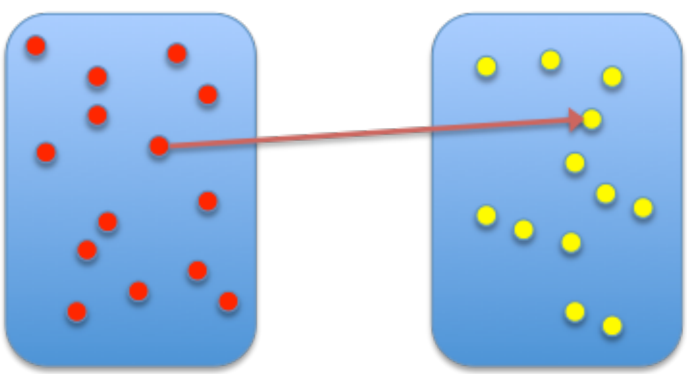
- Multi-objective (or multi-criteria) optimization
  - Discipline focused on solving multiobjective optimization problems (MOPs)

- Example: car trip between two cities
  - Objectives
    - Minimizing time
    - Minimizing fuel consumption
  - Decision variables:
    - Speed, instant consumption, ...



What is multi-objective optimization?

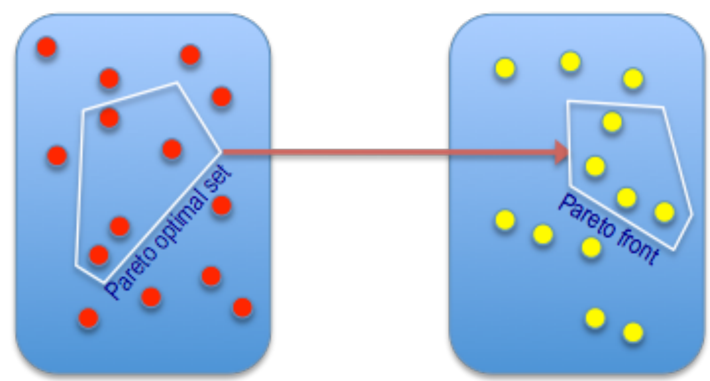
- In single-objective optimization (SO)
  - The optimal result is one single solution



X
(Solution space)

F(X)
(Objective space)

- In multi-objective optimization (MO)
  - The optimal result (Pareto optimal set) is a set of (non-dominated) solutions



X
(Solution space)

F(X), G(X), ...
Objective space

The dominance concept

- In single-objective optimization (SO)
  - We look for a single solution
  - The concept of "A better than B" is trivial

- In multi-objective optimization (MO)
  - We are not restricted to find a unique optimal solution
  - The concept of "A better than B" is not trivial

A	2	3	4	5
B	4	6	5	7

A is better than B

A	3	7	4	8
B	2	1	2	5

B is better than A

A	1	9	4	5
B	3	6	5	7

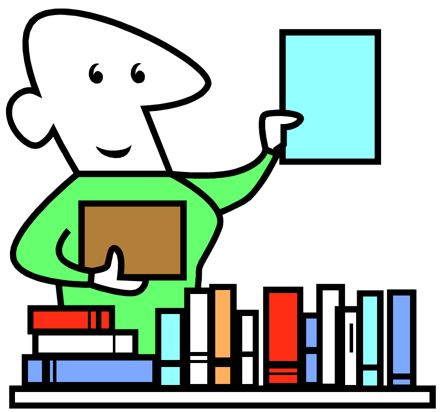
None is better

A and B are NON-DOMINATED

MO Optimization and Decision Making

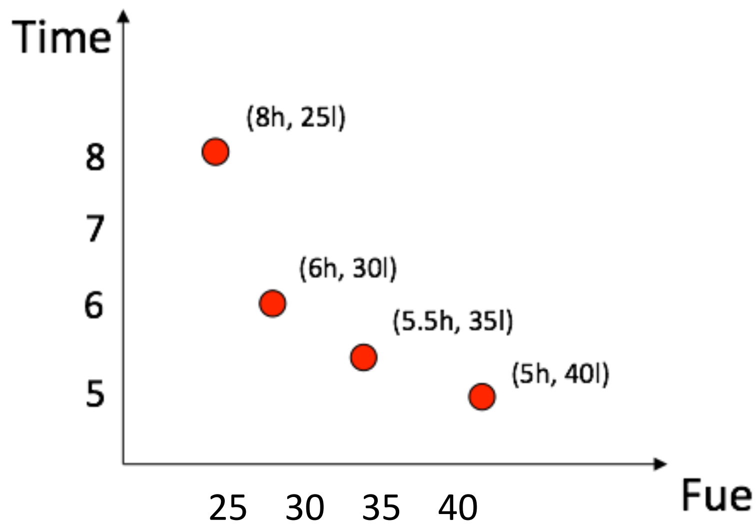
- Finding the Pareto front of a problem is not the last step in multi-objective optimization

- In practice, an expert in the domain (the decision maker) has to choose the best trade-off solution



MO Optimization and Decision Making

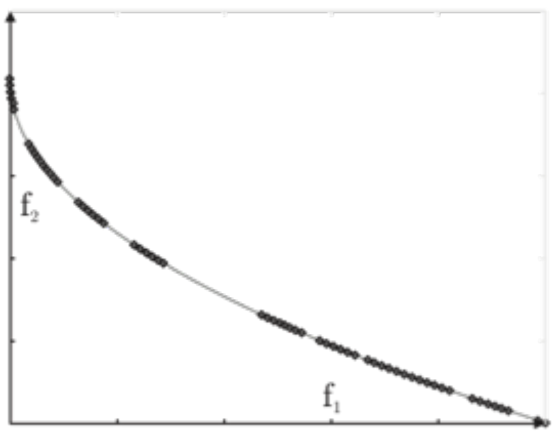
- In the example of the car trip
  - If time is important
    - Choose (5h, 40l)
  - If consumption is important:
    - Choose (8h, 20l)
  - Compromise solution:
    - (6h, 30l)



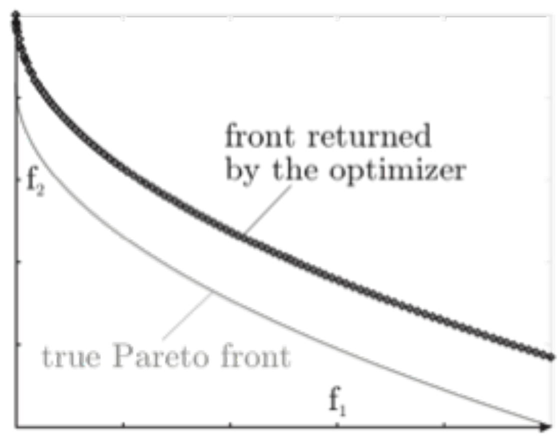
The Pareto Front

- The goal is to find the Pareto front
- Exact techniques are not useful in most cases
  - NP-hard complexity, non-linearity, epistasis , …
- Rely on approximation techniques
- Two key features to measure the quality of solutions
    - Convergence
    - Diversity

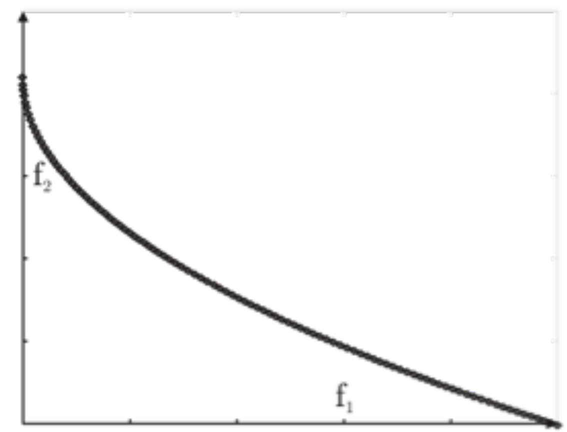
The Pareto Front



Bad diversity



Bad convergence



Ideal case

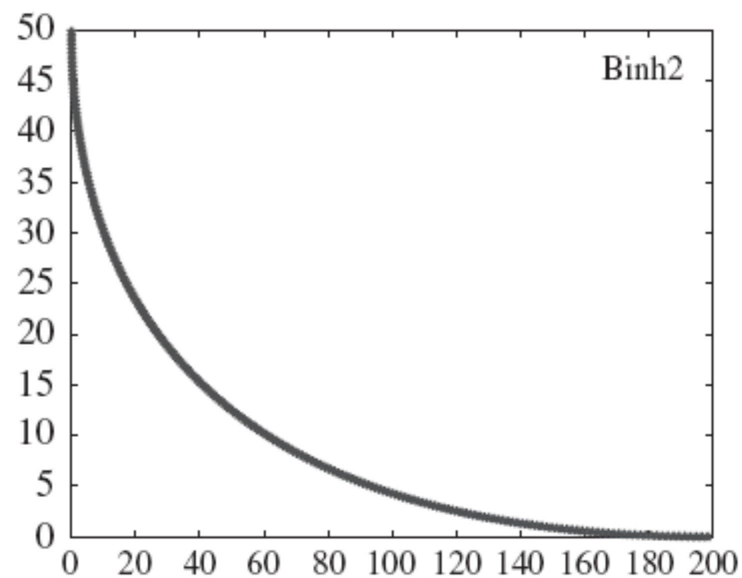
Pareto Front Example (I)

- Bi-objective problem

$$
\begin{aligned}
\text{Min } F &= (f_1(\vec{x}), f_2(\vec{x})) \\
f_1(\vec{x}) &= 4x_1^2 + 4x_2 \\
f_2(\vec{x}) &= (x_1 - 5)^2 + (x_2 - 5)^2
\end{aligned}
$$

Subject to:

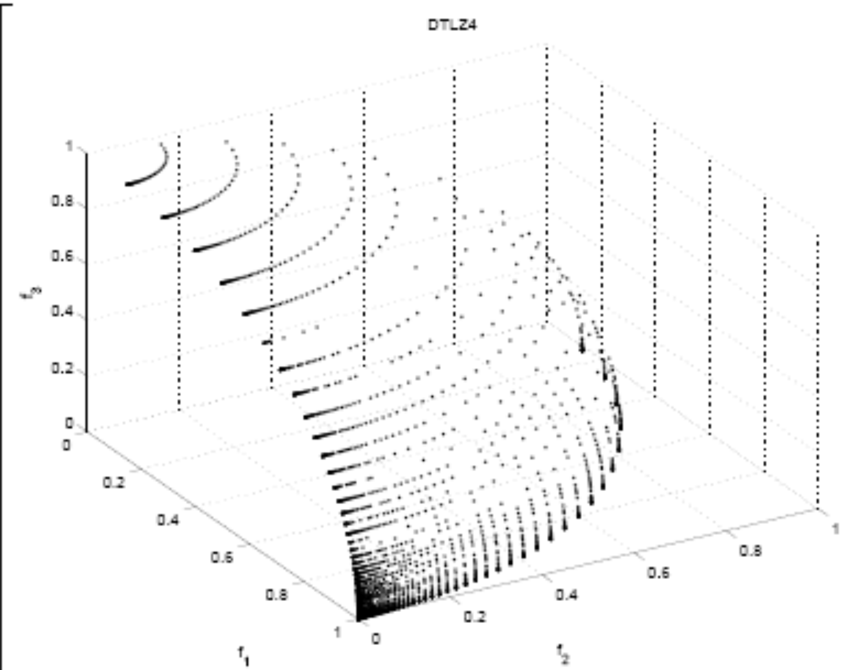
$$
\begin{array}{rcll}
g_1(\vec{x}) & = & (x_1 - 5)^2 + x_2^2 - 25 & \leq 0 \\
g_2(\vec{x}) & = & -(x_1 - 8)^2 - (x_2 + 3)^2 + 7.7 & \leq 0 \\
0 & \leq & x_1 & \leq 5 \\
0 & \geq & x_2 & \leq 3
\end{array}
$$



Pareto Front Example (II)

- Tri-objective problem

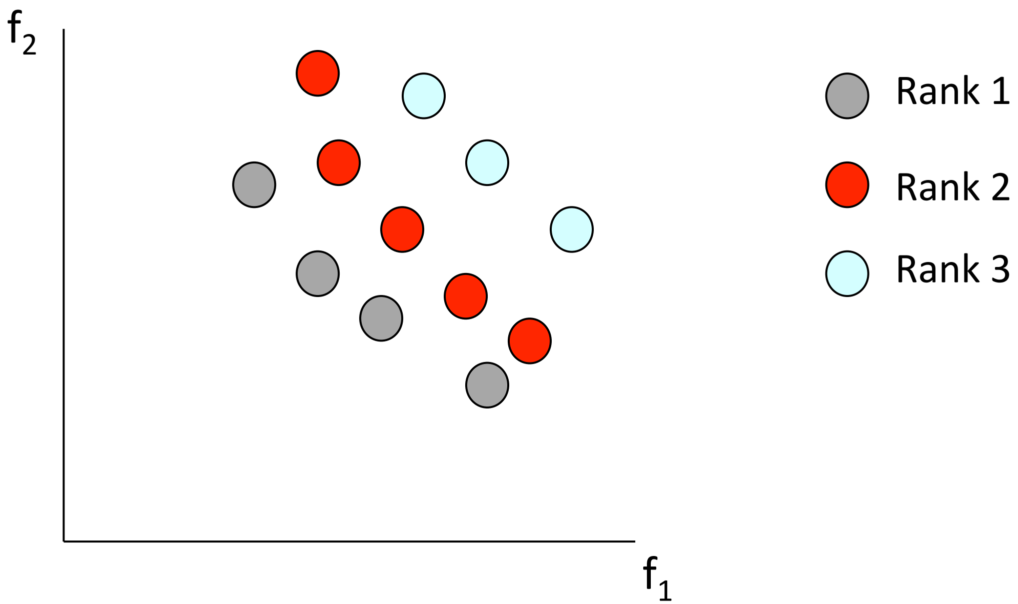
$$
\begin{aligned}
f_1(x) &= (1+g(x_M))\cos(x_1^\alpha \tfrac{\pi}{2})\cos(x_2^\alpha \tfrac{\pi}{2}) \\
f_2(x) &= (1+g(x_M))\cos(x_1^\alpha \tfrac{\pi}{2})\sin(x_2^\alpha \tfrac{\pi}{2}) \\
f_3(x) &= (1+g(x_M))\sin(x_1^\alpha \tfrac{\pi}{2}) \\
& \quad 0 \le x_i \le 1, \quad i = 1, 2, \ldots, n \\
g(x_M) &= \textstyle\sum_{x_i \in x_M} (x_i - 0.5)^2 \\
n &= 12 \\
x_M &= x_3, \ldots, x_{12} \\
\alpha &= 100
\end{aligned}
$$



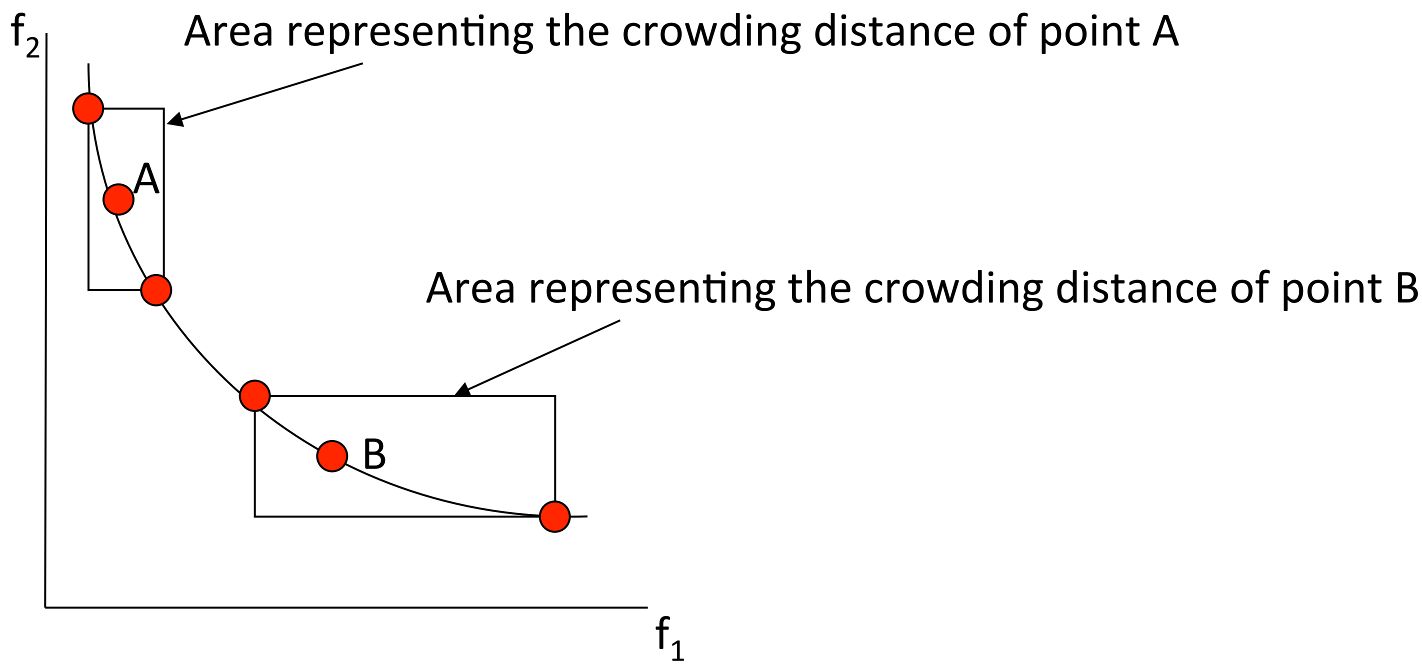
NSGAII Algorithm for MO Problems

- Non-dominated Sorting Genetic Algorithm
- Proposed by K. Deb (2002)
- The most popular metaheuristic for multi-objective optimization
- Features
  - Ranking using non-dominated sorting
  - Crowding distance as density estimator

NSGAII - Ranking

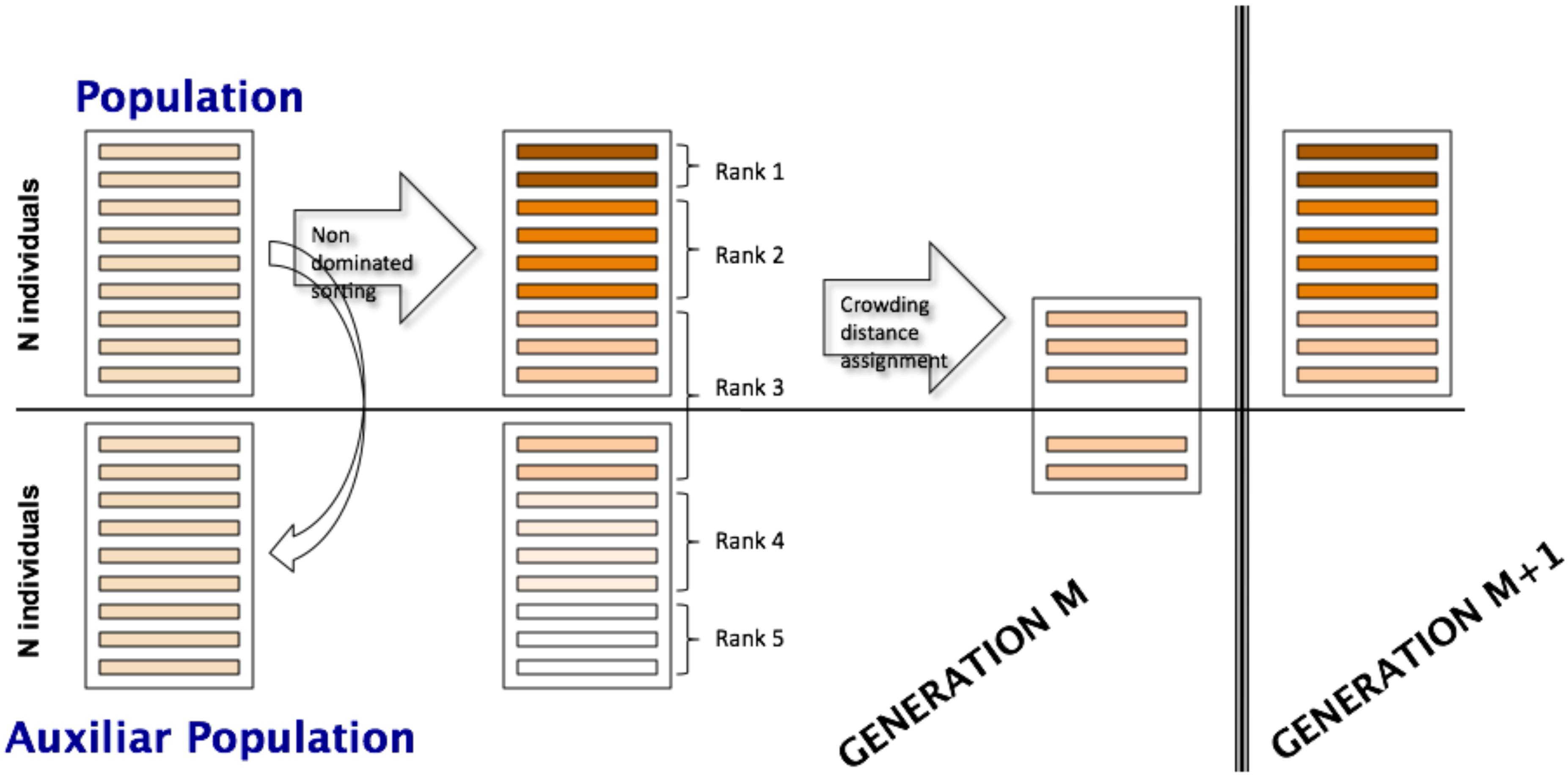


NSGAII - Crowding



Point B is in a less crowded region than point A

NSGAII Algorithm for MO Problems



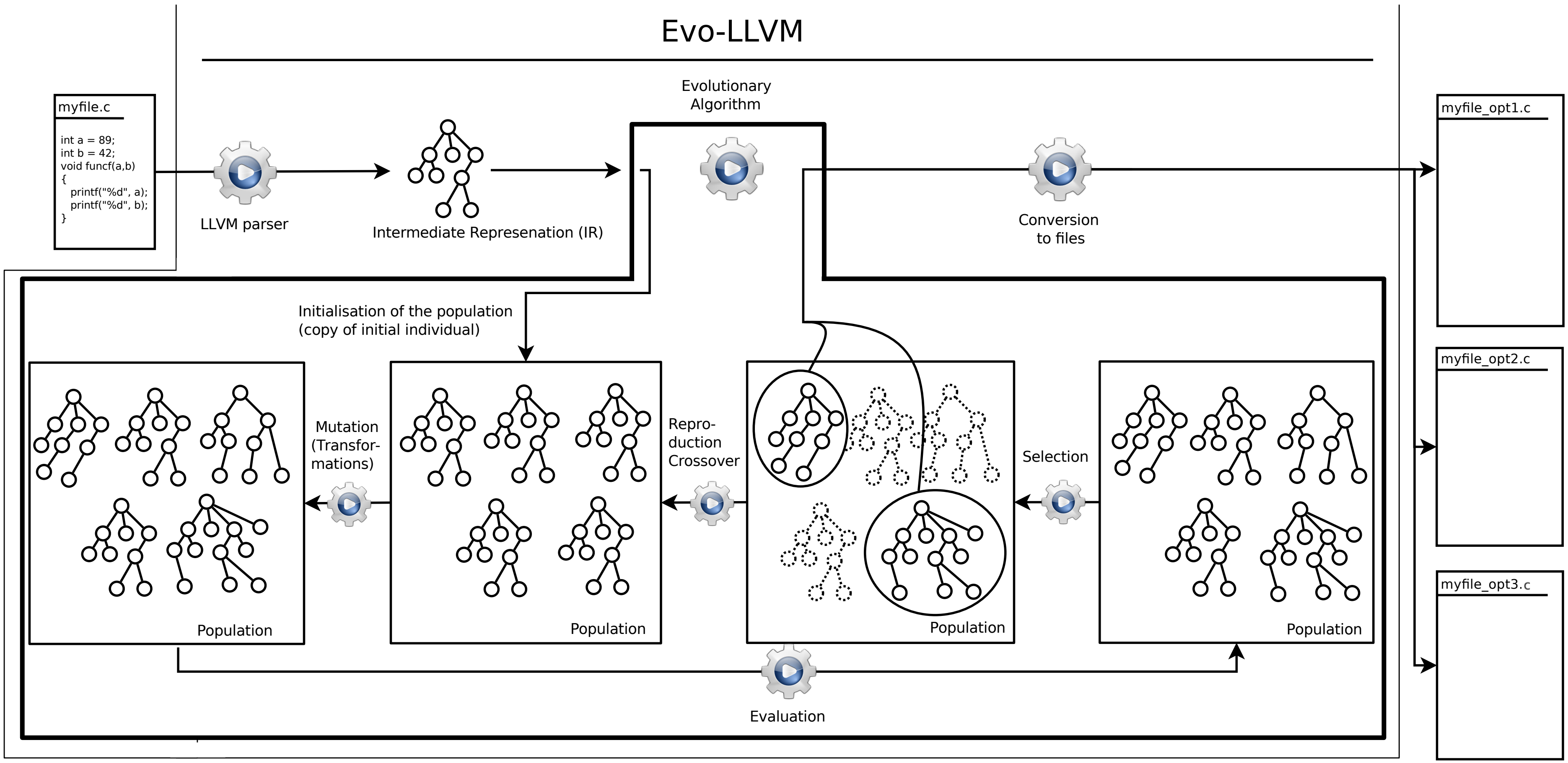
Outline

- Context and motivation
- The code optimization problem
- Introduction to multi-objective optimization
- The Evo-LLVM compiler framework
- Preliminary results
- Conclusion and perspectives

Evo-LLVM overview

- Exploit the flexibility offered by LLVM to manipulate the IR
- Take profit from applying a sequence of supported transforms
- Evaluate impact on (at least) two objectives:
  - Energy effciency of the produced executable
  - Run time
- Multi Objective Evolutionary Algorithms (MOEAs)
  - Build approximated Pareto-optimal solutions
  - In this work: NSGA-II

Evo-LLVM


Evo-LLVM
myfile.c
int a = 89;
int b = 42;
void funcf(a,b)
{
printf("%d", a);
printf("%d", b);
}
LLVM parser
Intermediate Represenation (IR)
Evolutionary Algorithm
Conversion to files
myfile_opt1.c
myfile_opt2.c
myfile_opt3.c
Initialisation of the population (copy of initial individual)
Mutation (Transfor-mations)
Repro-duction Crossover
Selection
Population
Population
Population
Population
Evaluation

Representation of solutions

- Given a **source program** P
- **Individuals** (I)
  - Composed by
    - LLVM byte code of P
    - Sequence of applied transforms
      - Variable length
  - Features
    - Semantically equivalent to P
    - Easily built from P

Parameters of NSGAII

- **Population size:** 50 individuals
- **Initial population:** Individuals are P with one random transformation
- **Mutations**: on each element of the sequence with prob. $P_m = 0.1$
  - Change the transformation by another randomly chosen one
  - Or append a new transformation
- **Cross-over**: Single-point cross-over
  - Limits the break of “good” sequences
- **Maximum number of generations:** 100

Benchmark

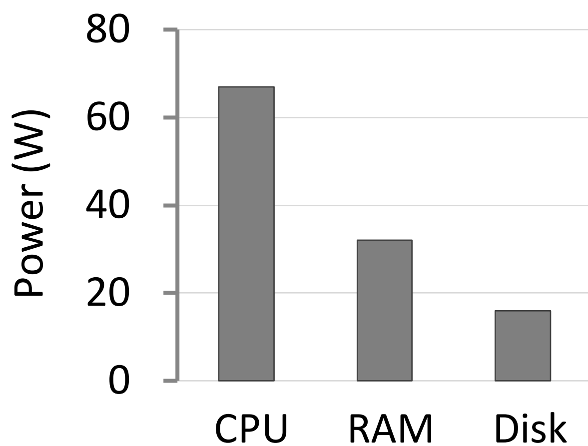
- **Quicksort** algorithm
  - Loops
  - Memory allocations
  - Recursion
  - Branching
- Test cases: strings of 100 and 1000 numbers
  - Random
  - Random, but with some duplicates
  - Random, but sorted: small-to-big
  - Random, but sorted: big-to-small

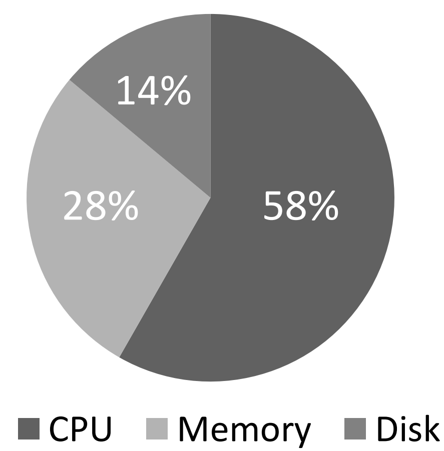
Fitness

- **Two** objectives
  - Execution time
    - Average runtime for each test-case
      - 100 runs
      - Sequentially executed
  - Power consumption
    - Average power consumption for each test-case
    - Power consumption based on estimations

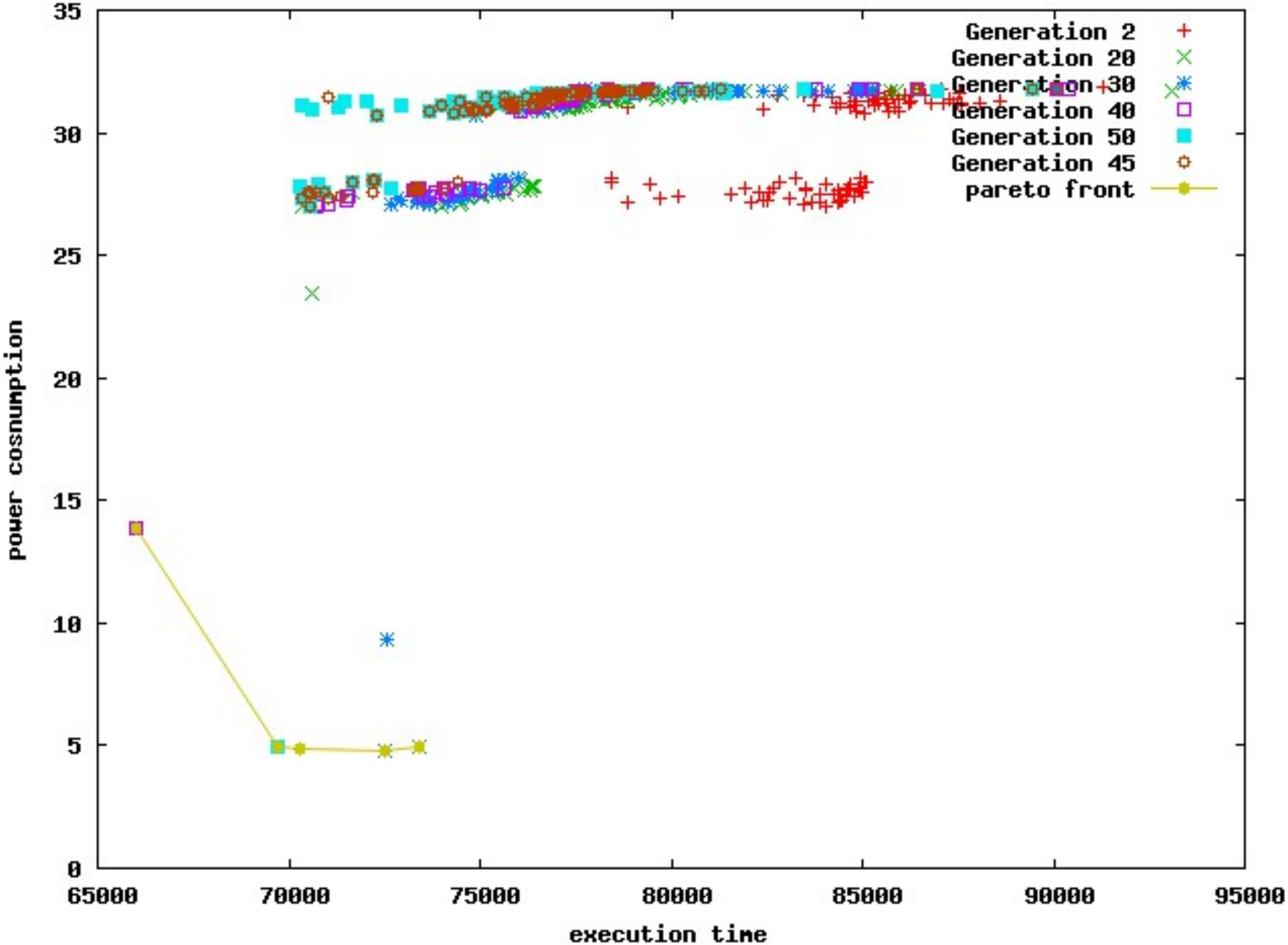
Estimation of Power Consumption

- Evaluated per evaluation process (i.e., per pid)
  - Based on ratio of the total power for 100 consecutive runs
  - Focus on **relative Avg.** CPU & memory usage per pid
    - /proc/<pid>/stat & /proc/<pid>/statm & /proc/meminfo
      Power(*pid*) = [0.58 X $\boldsymbol{\alpha}_{cpu}$(*pid*) + 0.28 X $\boldsymbol{\alpha}_{mem}$(*pid*)] P_{total}



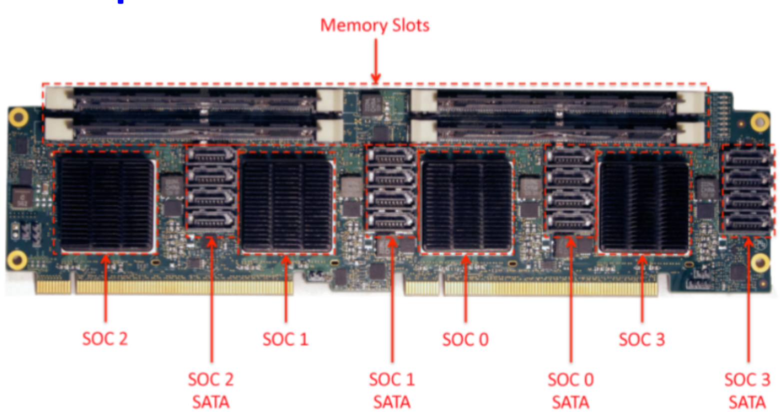


N. Kothari et al., Virtual Machine Power Metering and Provisioning, ACM Symposium on Cloud Computing, 2010


Generation 2
Generation 20
Generation 30
Generation 40
Generation 50
Generation 45
pareto front
power cosnumption
execution time
0
5
10
15
20
25
30
35
65000
70000
75000
80000
85000
90000
95000

Estimation of Power Consumption

- Option 1: Intelligent Platform Management Interface (IPMI)
  - Defines a set of interfaces for out-of-band management of computer systems
    - Connection to HW and not OS
  - Provides power measurement of the card



Calxeda EnergyCard module (4 ARM Cortex A9 processors)

- Option 2: Build high precision power metering device

Conclusions

- Evo-LLVM evolves a given source code to produce energy aware versions
  - Use MO to look for appropriate transformation sequences
  - Energy and performance metrics for fitness evaluation
  - Optimization is bound to a given computing system
- Preliminary experiments show promising results
  - Still, long way ahead
    - Need better energy monitoring
    - Improve experimental settings
    - Only applied to a pedagogical example (quicksort)

Thanks!

Bernabé Dorronsoro
University of Cadiz
bernabe.dorronsoro@uca.es
http://bernabe.dorronsoro.es