Universidade de Évora / LISP

Constraint Solving on Hybrid Systems

Pedro Roque, Vasco Pedro and Salvador Abreu

INAP 2017

Contents

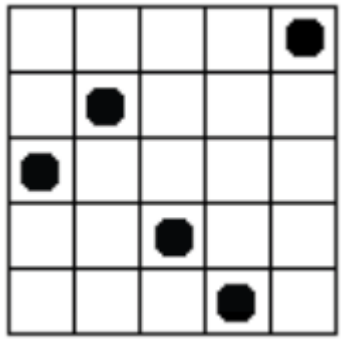
Next...

- Introduction
  - Constraint Satisfaction Problems (CSPs)
  - Objective
  - Challenges
- Solver architecture
  - Framework
  - Problem splitting
  - Load balancing
  - Communication
- Results and discussion
  - Testing environment
  - Benchmark suite
  - Results with PHACT
  - Comparison with other solvers
- Conclusion and future work
  - Conclusion
  - Future work

Constraint Satisfaction Problems (CSPs)

Constraint Satisfaction Problems (CSPs) allow representing many problems from different areas and are defined by a set of variables, their domains and constraints between them

- Costas Array [Diaz et al. 2012]
  - Placing a set of n dots on a n x n matrix
  - Each row and column contains only one dot
  - All vectors between dots are distinct



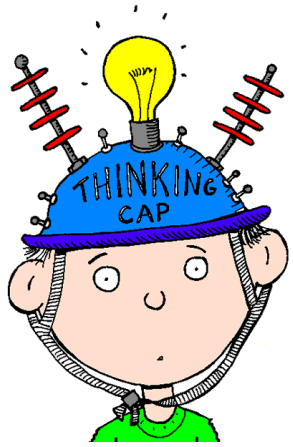
Objectives

Solve CSPs faster by splitting the work between multiple devices

- CPUs
- GPUs
- Intel Many Integrated Cores (MICs)
- Any other device compatible with OpenCL

Challenges

- Hardware architectural differences between devices
- Reduce to the minimum the amount and number of data transfers between devices
- While solving a CSP on a device, it is not possible to programmatically pause or stop that operation (OpenCL)
- Make all devices finish at the same time



Next...

- Introduction
  - Constraint Satisfaction Problems (CSPs)
  - Objective
  - Challenges
- Solver architecture
  - Framework
  - Problem splitting
  - Load balancing
  - Communication
- Results and discussion
  - Testing environment
  - Benchmark suite
  - Results with PHACT
  - Comparison with other solvers
- Conclusion and future work
  - Conclusion
  - Future work

Framework

- Parallel Heterogeneous Architecture Constraint Toolkit (PHACT) runs on
  - Linux and Windows
  - All devices compatible with OpenCL (CPUs, GPUs, MICs).

- Uses
  - One master process
  - One thread per device on the machine CPU
  - Multiple exploration threads on each device

Problem splitting

The problem is split into multiple disjoint sub-problems

- Up to a few millions, depending on the type and amount of devices
- Each device will receive sets of sub-problems to explore
- The size of each set depends on the device that will receive it
- Each exploration thread will explore a sub-problem at a time

Load balancing

- Each device receives two small sets of sub-problems
- When more than one device finishes exploring its two first sets, rank is calculated for those devices

$$rank(d) = \frac{\frac{1}{avg(d)}}{\sum_{i=1}^{m} \frac{1}{avg(i)}}, \quad avg(i) > 0$$

- The first device to finish exploring the first two sets will then receive twice the number of sub-problems

Load balancing

- After calculating rank for the first time, each device will get the number of sub-problems resulting from multiplying rank by 1/3 of the sub-problems that were not used for the calculation
- All devices will update their rank after solving each set of sub-problems
- Rank will be multiplied by the remaining sub-problems
- Repeat until all work is done

Device	Average time per sub-problem (ms)	Rank	Remaining sub-problems to explore	Size of next set of sub-problems
Device 1	0.00125	0.625	1,233,482	770,926
Device 2	0.00236	0.331	462,556	153,106
Device 3	0.01782	0.044	309,450	13,616

Communication

- To reduce the number and size of data transfers
  - The full CSP is passed to each device only once at the start of the solving process
  - Each set of sub-problems its represented only by an enumeration and information on how to generate each sub-problem

$$SS1 = \{V1 = \{1\}, V2 = \{1\}, V3 = \{1, 2\}\}$$
$$SS2 = \{V1 = \{1\}, V2 = \{2\}, V3 = \{1, 2\}\}$$
$$SS3 = \{V1 = \{2\}, V2 = \{1\}, V3 = \{1, 2\}\}$$
$$SS4 = \{V1 = \{2\}, V2 = \{2\}, V3 = \{1, 2\}\}$$

  - Each device only returns the solution or number of solutions found

Next...

- Introduction
  - Constraint Satisfaction Problems (CSPs)
  - Objective
  - Challenges
- Solver architecture
  - Framework
  - Problem splitting
  - Load balancing
  - Communication
- Results and discussion
  - Testing environment
  - Benchmark suite
  - Results with PHACT
  - Comparison with other solvers
- Conclusion and future work
  - Conclusion
  - Future work

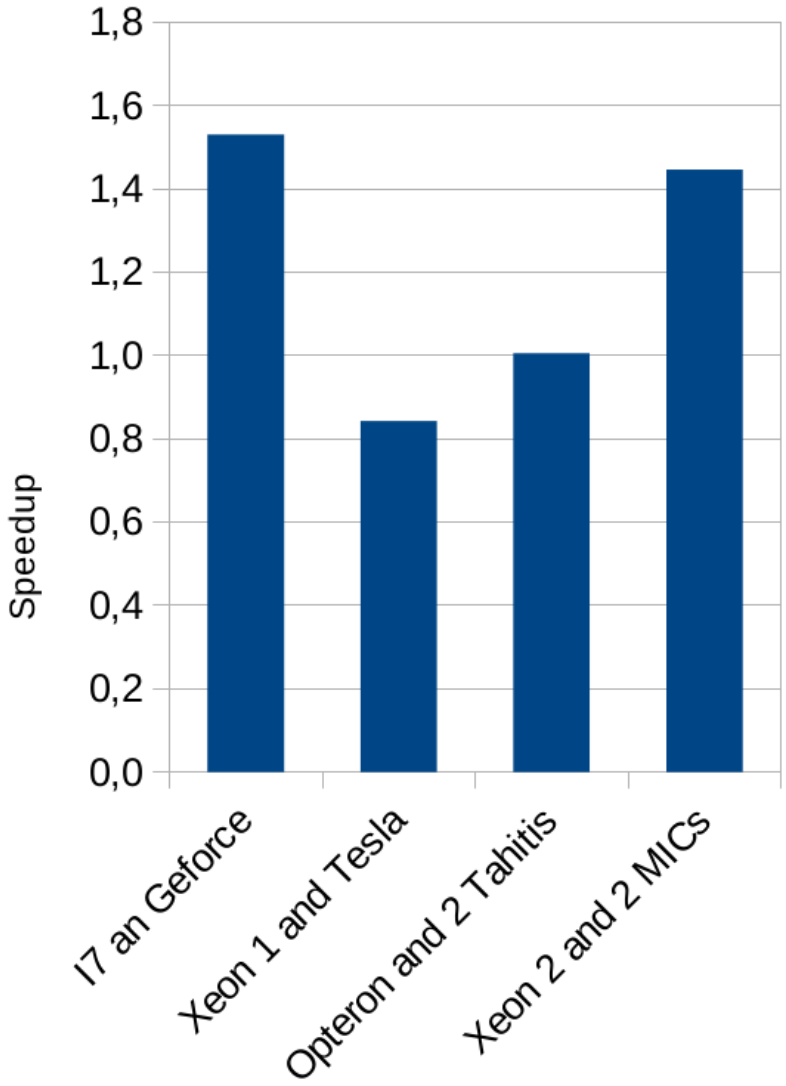
Testing environment

- PHACT was executed on Linux and on
  - Machine with 32 GB of RAM (referred to as M1) and
    - Intel Core i7-4870HQ (referred to as I7)
    - Nvidia GeForce GTX 980M (Geforce)
  - Machine with 64 GB of RAM (M2) and
    - Intel Xeon E5-2690 v2 (Xeon 1)
    - Nvidia Tesla K20c (Tesla)
  - Machine with 128 GB of RAM (M3) and
    - AMD Opteron 6376 (Opteron)
    - Two AMD Tahitis (Tahiti 1 and Tahiti 2)
  - Machine with 64 GB of RAM (M4) and
    - Intel Xeon CPU E5-2640 v2 (Xeon 2)
    - Two Intel Many Integrated Core 7120P (MIC 1 and MIC 2)

Benchmark suite

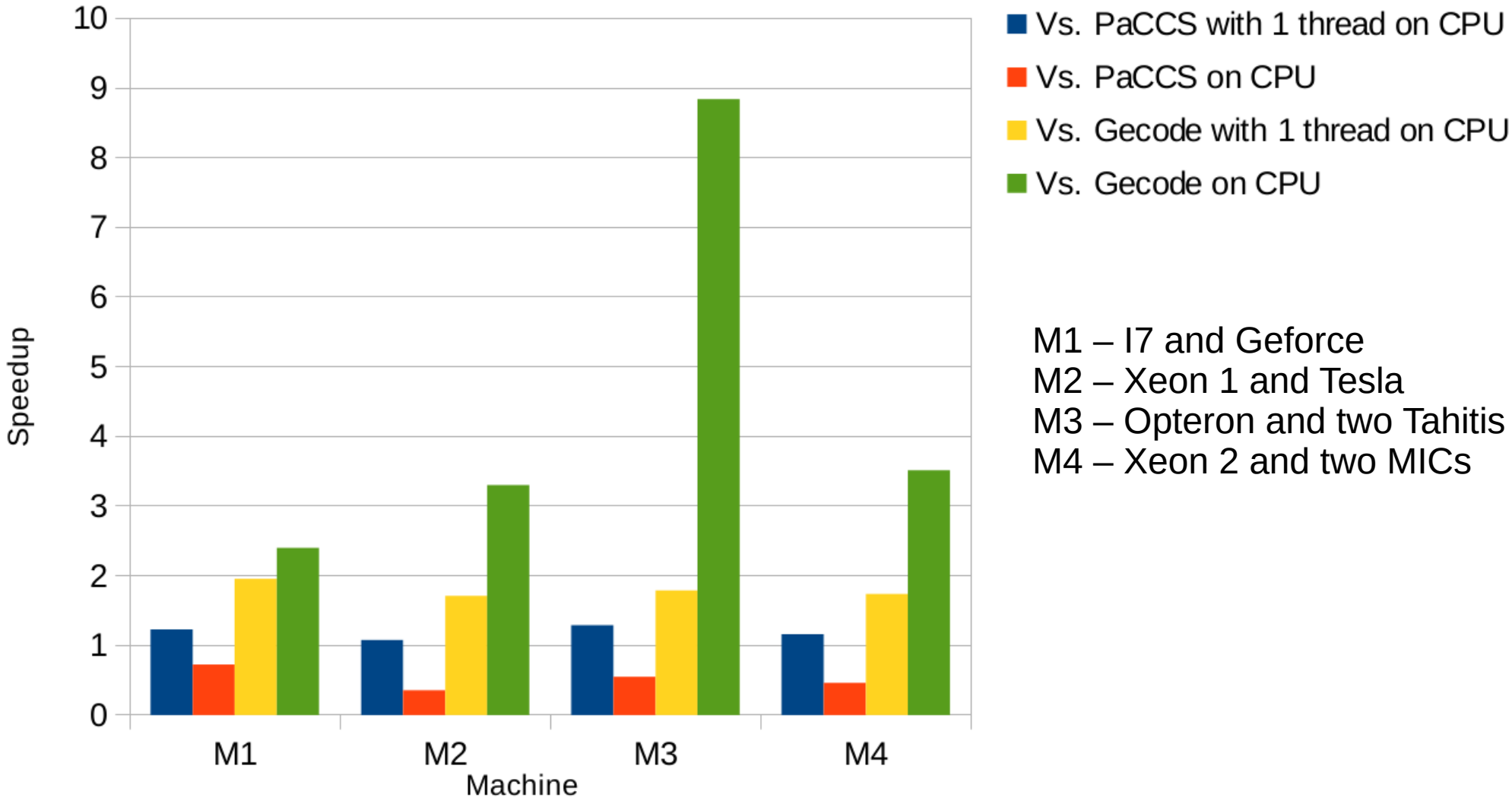
- 6 CSPs with 2 different dimensions each
  - All Interval, Costas Array, Golomb Ruler, Langford Numbers, Latin Squares and Market Split
- Counting all the solutions, optimizing and finding one solution
- Using
  - 1 CPU thread
  - All the CPU threads
  - The CPU and one device
  - The CPU and two devices

Speedups with more devices



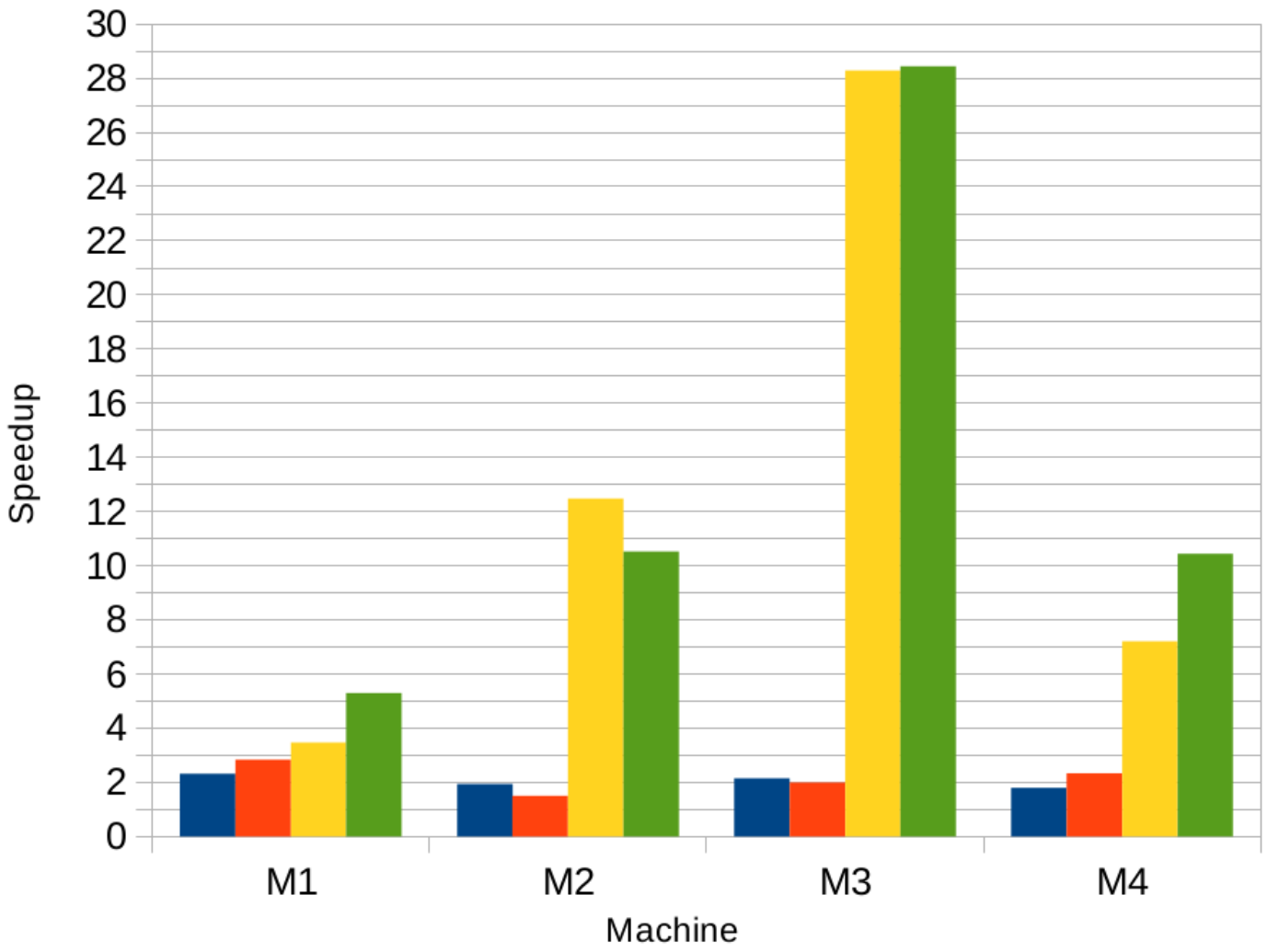
- Older GPUs are not so efficient for backtracking and constraint propagation
- The amount of total work increases as the number of devices also increases
  - More sub-problems are generated
  - More threads for controlling the devices
- Optimization and search for one solution may lead to unnecessary work
- Load balancing after depleting the set
- Load balancing at finishing times

Comparison with PaCCS and Gecode



M1 – I7 and Geforce
M2 – Xeon 1 and Tesla
M3 – Opteron and two Tahitis
M4 – Xeon 2 and two MICs

Comparison with PaCCS and Gecode



M1 – I7 and Geforce
M2 – Xeon 1 and Tesla
M3 – Opteron and two Tahitis
M4 – Xeon 2 and two MICs

Next...

- Introduction
  - Constraint Satisfaction Problems (CSPs)
  - Objective
  - Challenges
- Solver architecture
  - Framework
  - Problem splitting
  - Load balancing
  - Communication
- Results and discussion
  - Testing environment
  - Benchmark suite
  - Results with PHACT
  - Comparison with other solvers
- Conclusion and future work
  - Conclusion
  - Future work

Conclusion

- To our knowledge, PHACT is the only existent constraint solver capable of using CPUs, GPUs and MICs
- Using other devices to help the machine CPU allowed to achieve speedups of up to 4.66
- When using all the devices it achieved a (geometric) mean speedup of 2.11 when compared with PaCCS and 11.34 comparing with Gecode
- Load balancing at finishing times can yet be improved between devices and inside each device
- Synchronization between devices must be more frequent when optimizing and when finding one solution

Future work

- Improve synchronization between devices at finishing times
- Improve synchronization between each device threads at finishing times
- Extend PHACT to
  - Allow the input of FlatZinc models
  - To work on distributed environments

Questions?