Massively Parallel Resolution of Combinatorial Problems on MultiGPU Clusters

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE — CReSTIC — Centre de Calcul ROMEO, MAISON DE LA SIMULATION de Champagne-Ardenne

J Loiseau, Ch Jaillet, F Alin, M Krajecki

University of Reims Champagne-Ardenne FRANCE + CReSTIC

NVIDIA CUDA RESEARCH CENTER

Motivations

Solve combinatorial problems on mutli-GPU clusters
- Choose most appropriate library and technologies
- Use the whole computation power

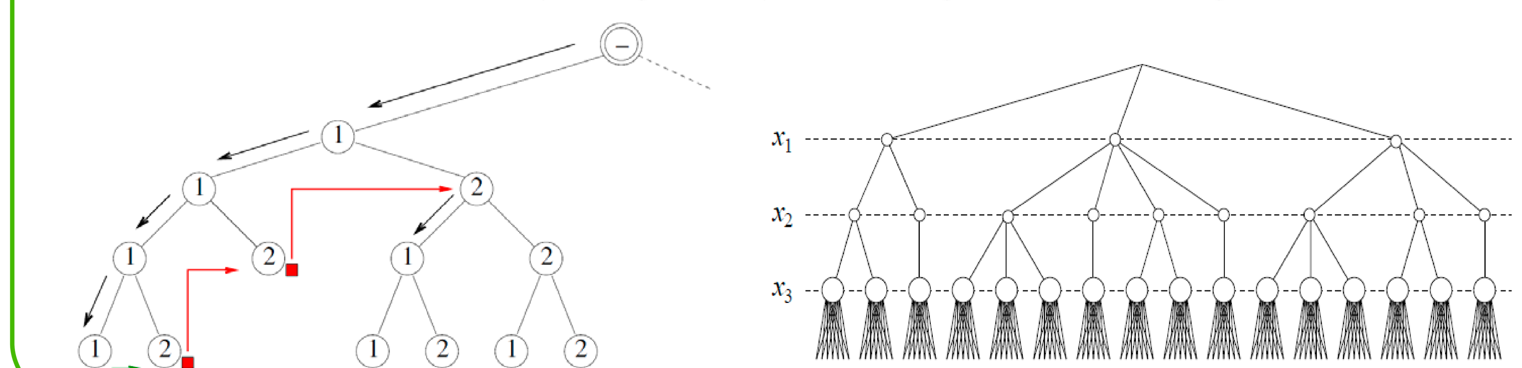
Optimize CUDA threads placement with blocks, grids and streams
- Use NVIDIA tools to optimize the resolution and Cuda code

Langford Problem as a Benchmark
- Huge combinatorial explosion
- Challenge for the next record => *L(2,27)*

Combinatorial Problems Generic Representation

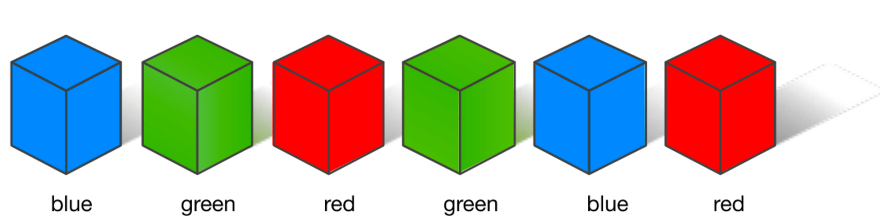
CSP = Constraint Satisfaction Problem:
- Consist of a set of variables, a domain of values for each variable and a set of constraints
- 3-uple (X,D,C)

Combinatorial problems representation:
- NP-complete => SAT formalism => CSP formalism
- Resolution method (our choice)
  - Tree representation with a static order of the variables and of their values
  - Backtracking traversal of the search tree
- Parallel resolution
  - Tasks generation = search space partitioning
  - FIIT : *Finite number of Independent and Irregular Tasks*
  - Our choice : develop to a given depth, avoiding inconsistent assigments



Langford Problem

Origin: C Dudley Langford observed his son playing with colored cubes, and he noticed a singular cubes arrangement :
- two cubes of a given color separated by 1 other cube
- two cubes of another color separated by 2 cubes
- the last two cubes separated by 3 other cubes



Description: *n* pairs of cubes => count the arrangements such that the distances between the two cubes of the different pairs are 1,2, … *n*

L(2,n) represents the number of these solutions, up to reversal

Formalism: a Langford sequence of order *n* is composed of *2n* integers, $L_1, \dots L_{2n}$

$$\text{such that } \begin{cases} \forall\, i \in [\![1,2n]\!] \; L_i \in [1,n] \\ \forall\, k \in [\![1,n]\!] \; \exists!\,(i,j),\, i<j,\; L_i = L_j = k \\ \forall\, k \in [\![1,n]\!] \; L_i = L_j = k,\; i<j \Rightarrow j-i = k+1 \end{cases}$$

Some results: *L(2,n) ≠ 0 ⇔ n = 4k or n = 4k-1, k > 0*

n	*L(2,n)*	method
…	…	Miller algorithm
15	39,809,640	
16	326,721,800	
19	256,814,891,280	2.5 y (1999) DEC Alpha
		Godfrey algorithm
20	2,636,337,861,200	1 w (2002) AMD/Pentium
23	3,799,455,942,515,488	distributed, 4 d (2004) ≈70 processors (our team)
24	46,845,158,056,515,936	distributed, 3 m (2004) ≈12 processors (our team)

Generic Distribution and Computation Scheme, with GPU Regularized Tree Traversal

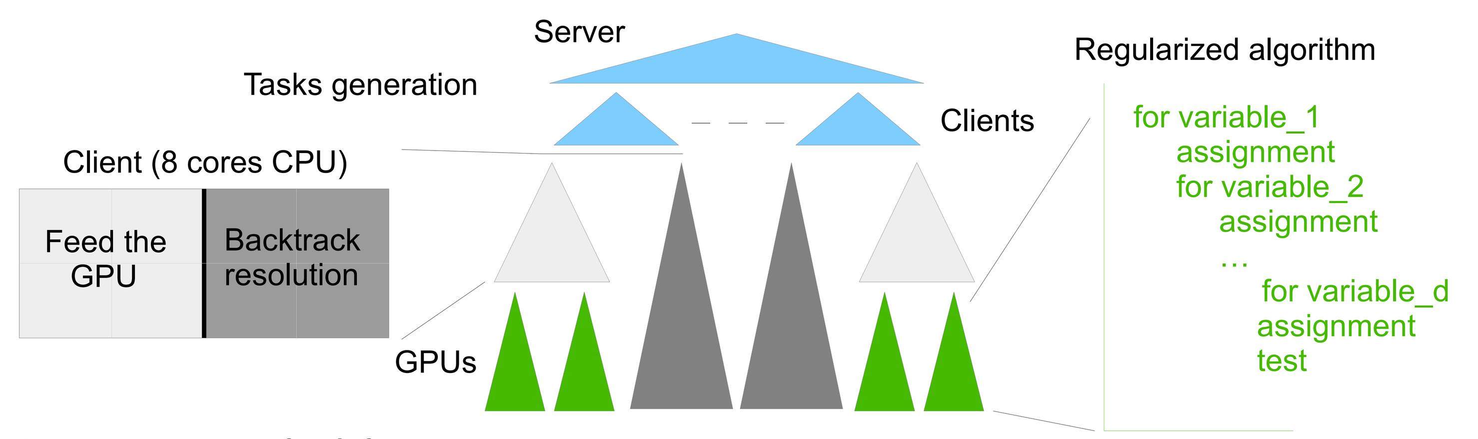
Cluster view:
- Cluster as distributed nodes aggregation
  - Client-Server repartition of the tasks
- Nodes : GPU coupled with CPU

Multi-level parallelization
- *MPI* communications
- *OpenMP* cores repartition
- *CUDA* K20X GPUs

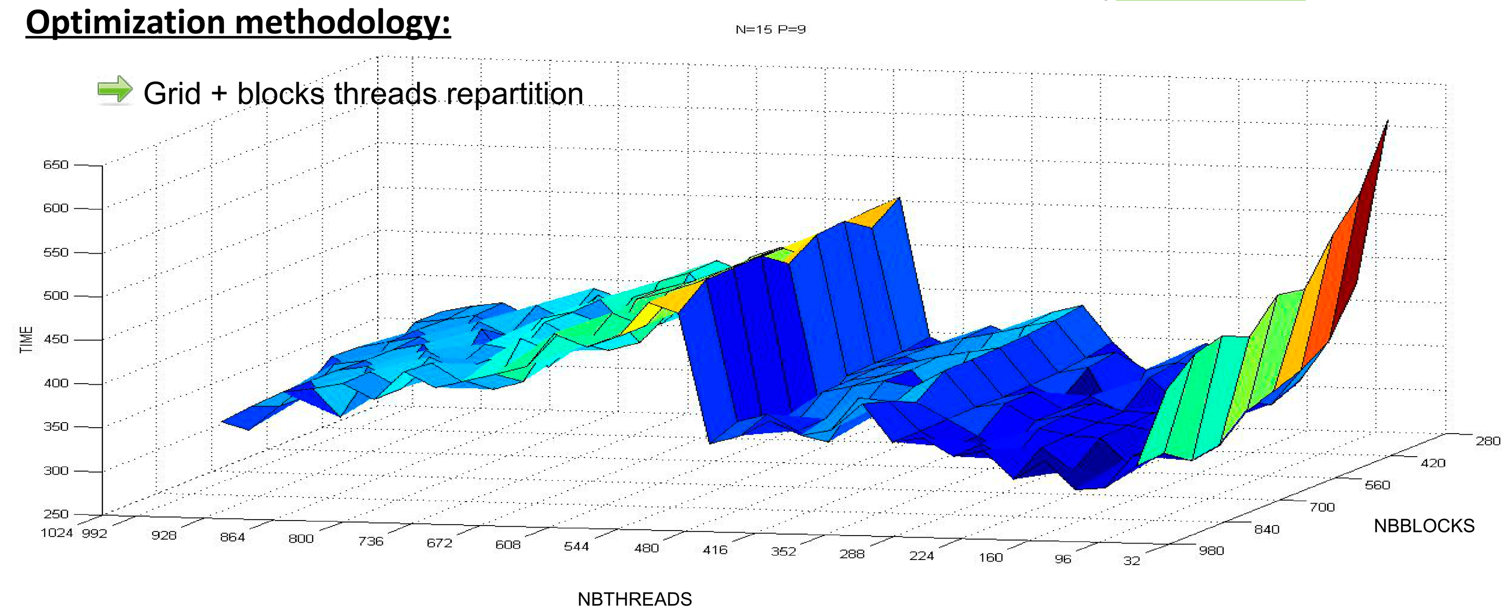
Problem: GPUs suffer from threads divergence when branching

Proposition: Create regularized GPU tasks
- Generate tasks to a given level and prepare work for GPUs
- GPUs tasks are vectorized : exhaustive computation of the last few levels

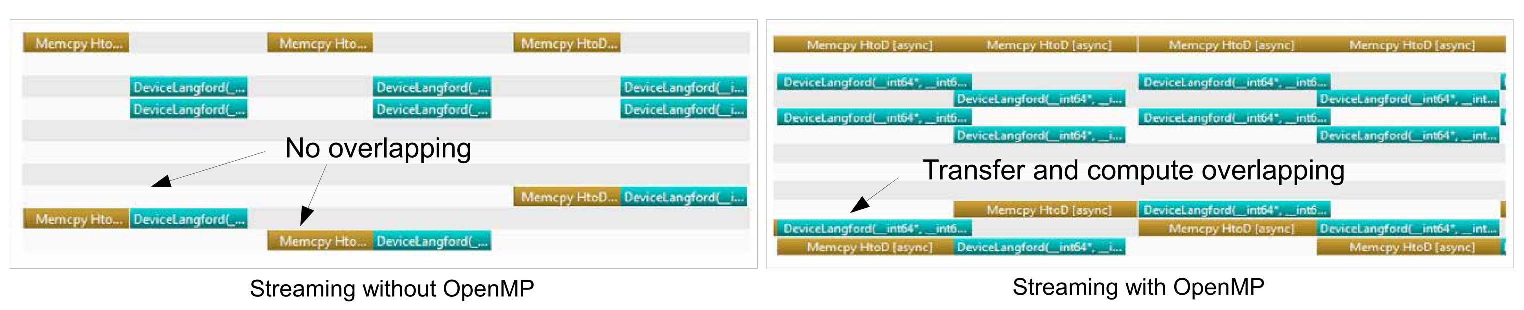
Method:
- A task is a consistent arrangement of placed pairs represented by a mask that reports the free places
- Each part place a defined number of pairs and generate binary representation called ***masks***



Optimization methodology:
- Grid + blocks threads repartition



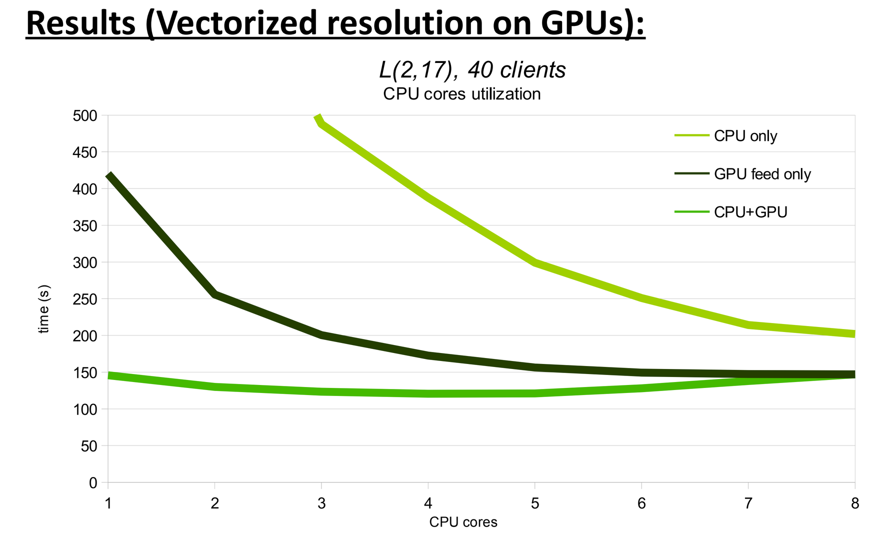
- Limitation due to the number of registers: 5 pairs on GPU => 57 registers/threads
- Using streams for the memory/computation overlapping

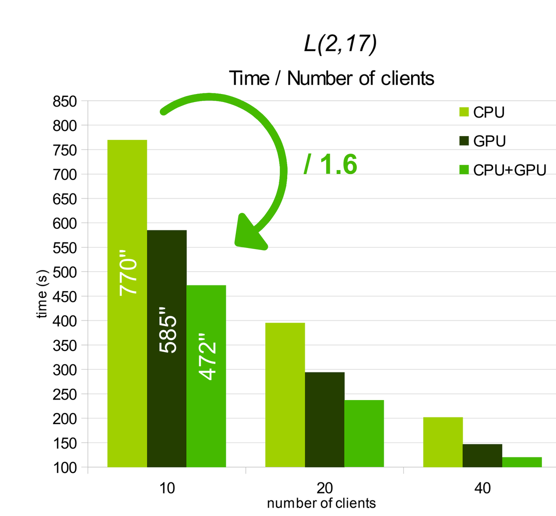


Streaming without OpenMP — Streaming with OpenMP

- Adding OpenMP to generate data for the GPU

Results (Vectorized resolution on GPUs):





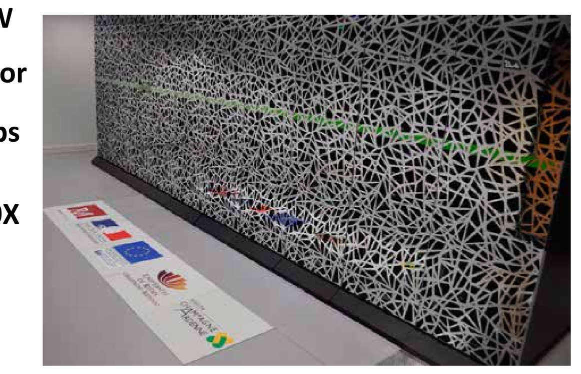
- 1.6 times faster however 2.10^5 times more nodes are traversed

Romeo HPC Tesla Cluster

Computing — Displaying

- THE GREEN 500: 5th 3131 MFLOPS/W — Bull Cool Cabinet Door
- TOP500: 151th 254.9 Tflops Linpack
- 260 NVIDIA Tesla **K20X** accelerators
- Bull: **130 Bull servers** — bullx R421 E3 Bull AE & MPI

260 INTEL Ivy Bridge E5-2650 v2 Processor, non-blocking **Mellanox Infiniband**, Slurm, 88 To Lustre (NetApp), 57 To home, 100 To Storage



Big Data, on-demand and remote

VirtualGL technology servers Quadro 6000 & 5800

NVIDIA GRID + Citrix Virtualisation **NVIDIA VGX K2**

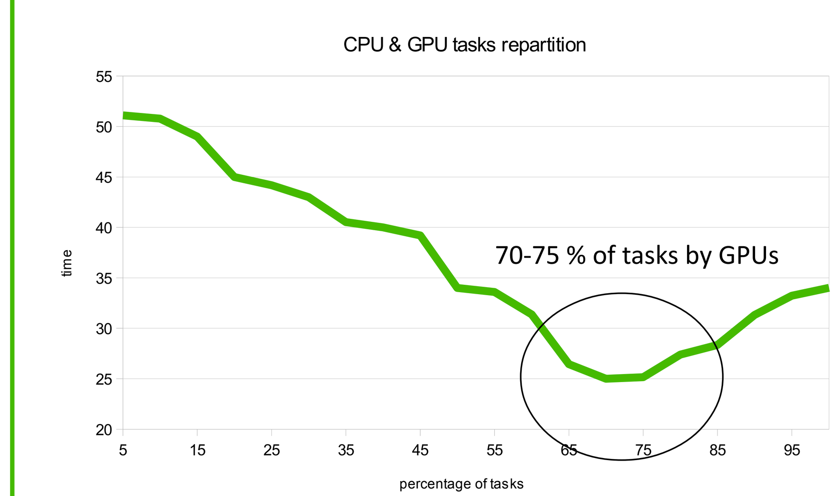
Scalable Graphics 3D cloud solution **NVIDIA K6000**

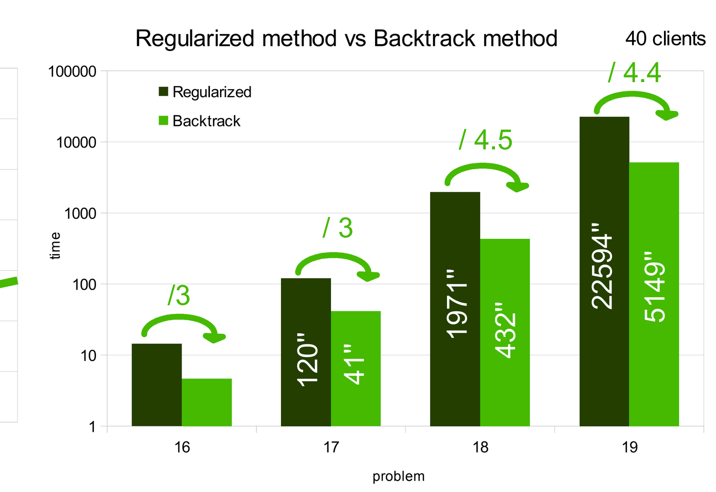
Massive Hybrid Backtrack Scheme for the Langford Problem : experimental results

Cluster view:
- Same representation as the Vectorized method
  - Client-Server repartition

Method: Miller algorithm
use Backtrack directly on GPU and don't care about lost of threads synchronisation
- Generate tasks to a given level and generate work for CPU/GPU
- Tasks spread between CPU cores and GPUs (static repartition)

Results (Backtrack only with trust in GPU scheduler):





GPU scheduler efficiency + scalable resolution scheme up to 100% of the ROMEO supercomputer

1 server + 128 nodes for solving => 256 CPU+GPU clients

n	time (seconds)
16	6.506
17	29.847
18	290.052
19	3197.526
20	39479.630 ≅ 11h
21	118512.420 ≅ 33h

Previous limit of the Miller algorithm was L(2,19) and took 2.5y computation

Backtack method limit now exceeded with: L(2,20) L(2,21)

Conclusions & Prospects

A massive parallel resolution scheme for combinatorial problems
- Generic resolution scheme
- Multi-level CPU-GPU parallelization
- GPU vectorized resolution *vs* native backtrack
- Further work: adapt to combinatorial optimization problems

Langford benchmark
- Proof of the resolution scheme
- *L(2,21)*: previous Miller algorithm limit exceeded
- New ways with the algrebraic Godfrey's method:
  - + natively regular; faster
  - - high GPU porting effort

Benchmark proposal for hybrid HPC cluster architectures

The exascale objective requires new hardware and algorithmic approaches. In this context the *Linpack* is disputed and new benchmarks should be proposed.

Some benchmark as the *Graph500* are becoming increasingly important. In a more general way, combinatorial problems should represent new benchmarks for HPC architectures. We aim at proposing new ways to solve them efficiently.