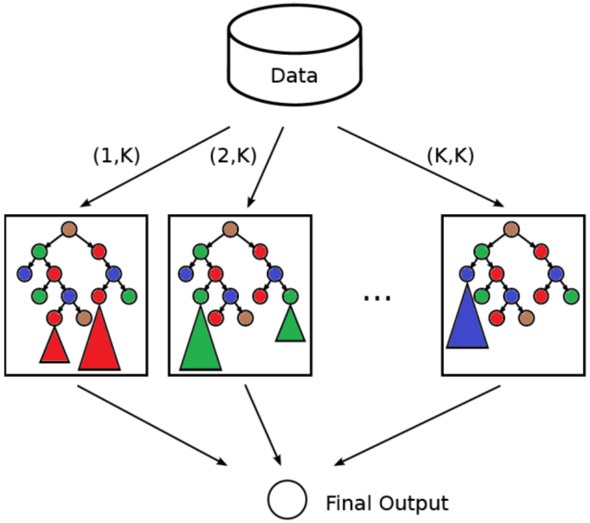
SelfSplit

automatic workload distribution in parallel computation

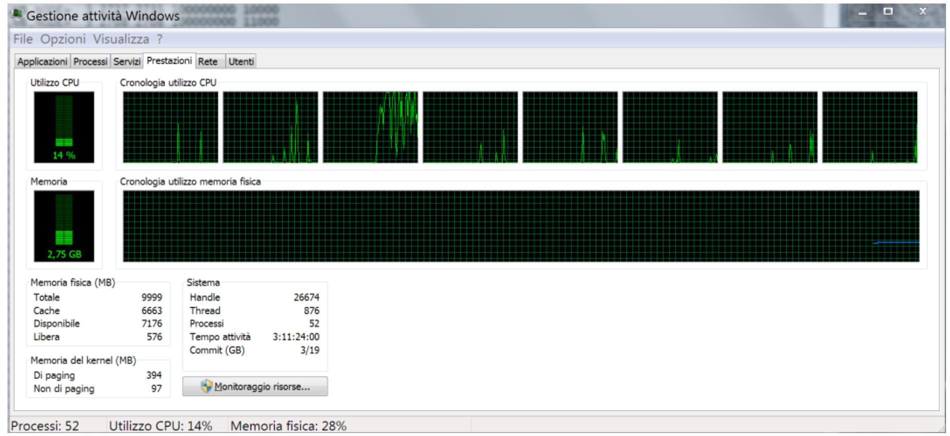
Matteo Fischetti, Michele Monaci, Domenico Salvagnin

University of Padova

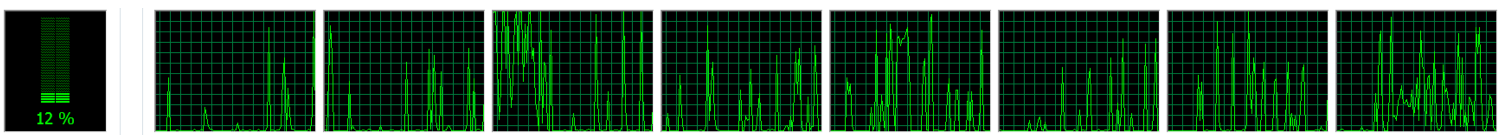


Parallel computation

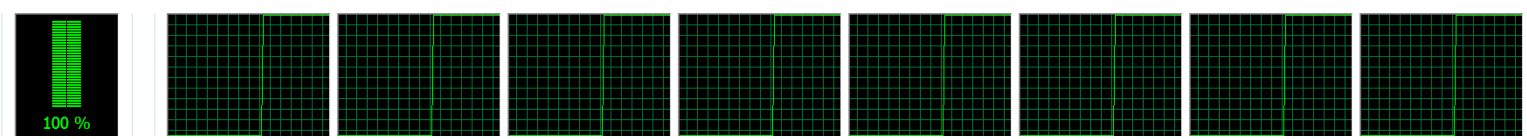
- Modern PCs / notebooks have several processing units (cores) available



- Running a sequential code on 8 cores only uses 12% of the available power…



- … whereas one would of course aim at using 100% of it



Distributed computation

- Affordable **servers** offer 24+ quadcore units (blades)

- **Grids** of 1000+ computers are available worldwide

- No doubt that **parallel computing** is becoming a must for CPU intensive applications, including optimization

- However, **many optimization codes are still sequential**…

Parallelization of a sequential code

- We are given a **deterministic sequential** source code based on a divide-and-conquer algorithm (e.g., **tree search**)

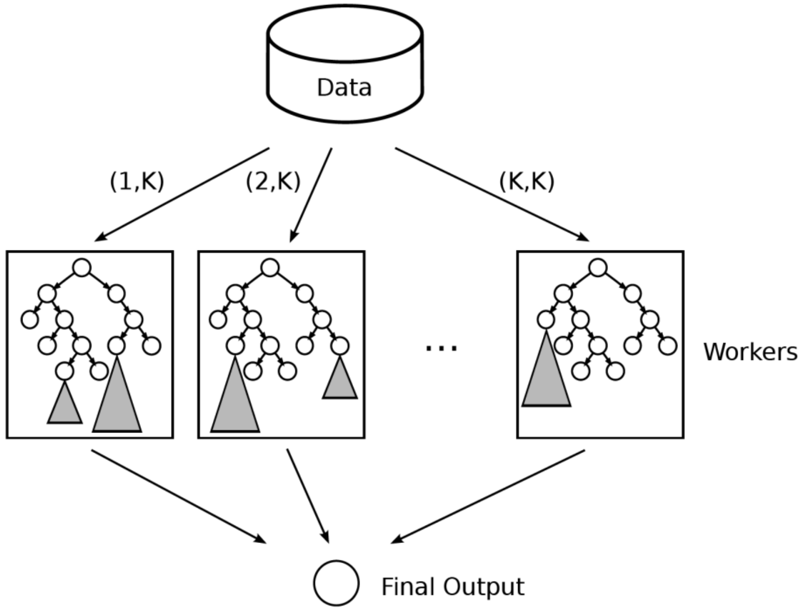
- We want to **slightly modify** it to exploit a given set of *K* (say) processors called **workers**

- **IDEA:** just run *K* times the **same sequential code** on the *K* workers

- … but modify the source code so as to **just skip some nodes** (that will be processed instead by one of the other workers…)

"Workload automatically splits itself among the workers"

SelfSplit

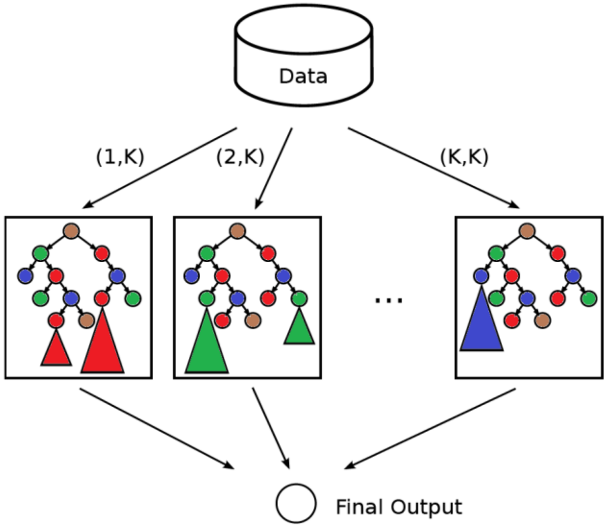


- Each worker reads the original input data and receives an **additional input** pair *(k,K)*, where *K* is the total number of workers and *k=1,…,K* identifies the current worker

- The same deterministic computation is initially performed, in parallel, by all workers (**sampling phase**), without any communication

- When enough open nodes have been generated, each worker applies a **deterministic rule** to identify and solve the nodes that belong to it (gray subtrees in the figure), without any redundancy. No (or very little) communication is required in this stage

Vanilla implementation



1. Two integer parameters (k, K) are added to the original input
2. A global flag **ON_SAMPLING** is introduced and initialized to true. The flag becomes false when there are enough open nodes in the branch-and-bound tree.
3. Each time a node n is created, it is deterministically assigned a *color* $c(n)$ which is a pseudo-random integer in $\{1, \cdots, K\}$ during the sampling phase, and $c(n) = k$ otherwise.
4. Whenever the modified algorithm is about to process a node n, condition

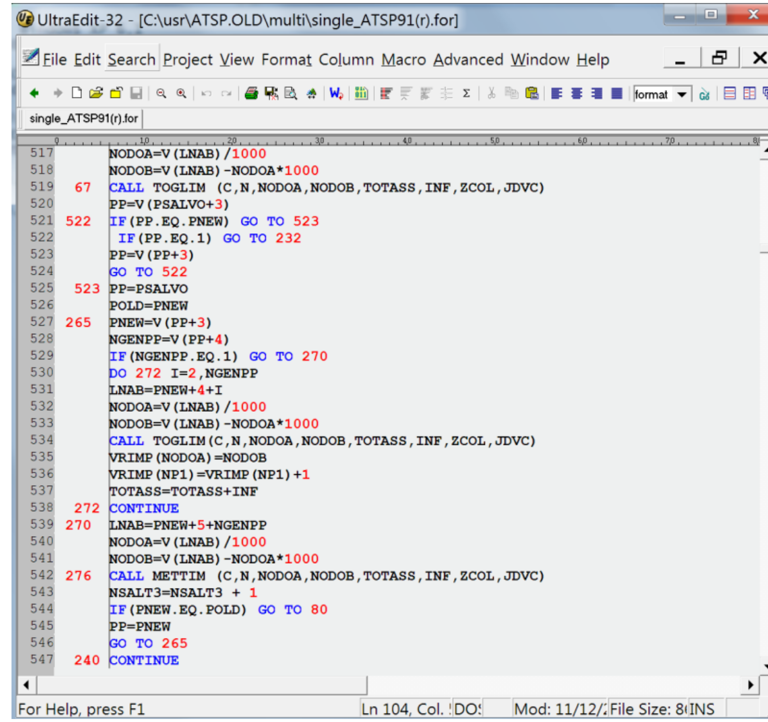
   $$(\textbf{not ON\_SAMPLING}) \quad \textbf{and} \quad (c(n) \neq k)$$

   is evaluated. If the condition evaluates to true, node n is just **discarded**, as it corresponds to a subproblem assigned to a different worker; otherwise, the processing of node n continues as usual and no modified action takes place.

Case study: ATSP B&B

Sequential code to parallelize: an old FORTRAN code of 3000+ lines from

M. Fischetti, P. Toth, “An Additive Bounding Procedure for the Asymmetric Travelling Salesman Problem”, Mathematical Programming A 53, 173-197, 1992.

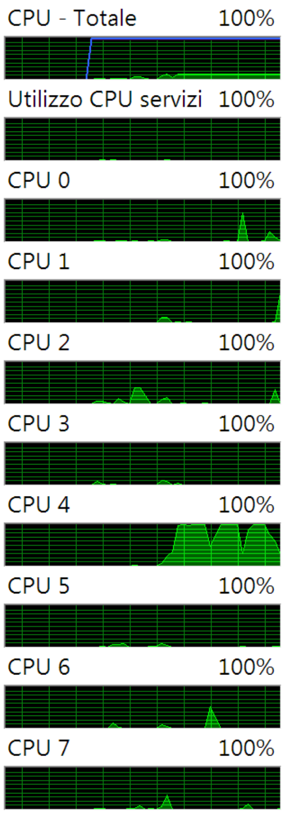


- Parametrized AP relaxation (no LP)
- Branching on subtours
- Best-bound first

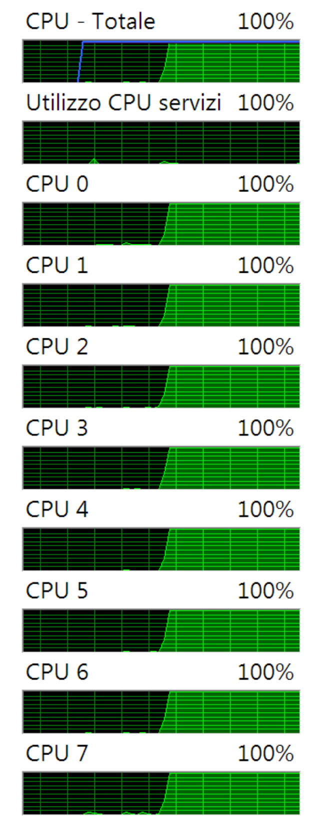
Vanilla SelfSplit: two variants

1. Absolutely **no communication** among workers (just **8 new lines** of code added to the sequential original code)

2. The value of the overall best incumbent is periodically written/updated on a single global file; each worker periodically reads it and only uses to possibly **<u>abort its own run</u>** (no other use allowed → overall method is still deterministic; **8+46 new lines** added)



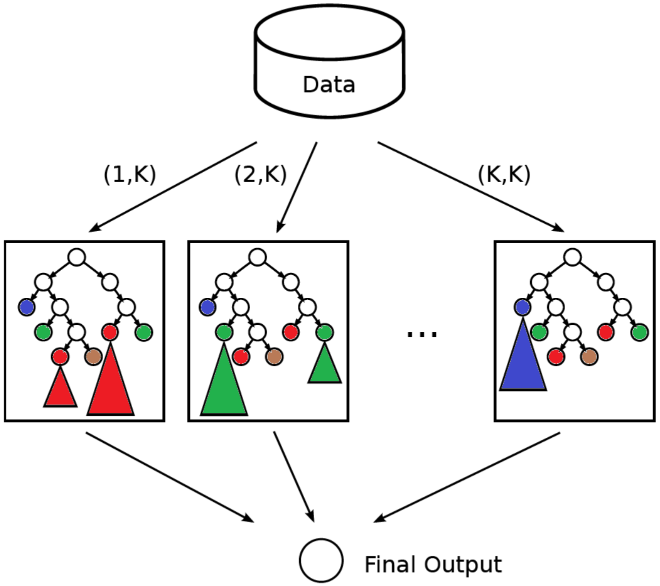
Results with 8 workers



worker	n.workers	value	elapsed (s.)	numnod
1	8	******	255	88917
2	8	******	267	94165
3	8	******	265	92989
4	8	4106	276	119573
5	8	4105	254	108105
6	8	******	264	93005
7	8	4104	264	104009
8	8	******	264	92253
1	1	4104	2411	708098

- Random instances taking 40 to 6,000 sec.s in sequential mode
- Version 2 (incumbent on file), **8 simultaneous runs on the same PC**
- Average speedup of **6.48** (geom.mean **5.47**) with 8 workers
- Speedup of **7+** for the most difficult instances

Paused-node implementation



1. As before, input parameters (k, K) are added.

2. Whenever the modified algorithm is about to process a node n, a boolean function **NODE_PAUSE**(n) is called: if true is returned, node n is just **paused** and the next node is considered.

3. When there are no nodes left to process, the *sampling phase* ends. All paused nodes, if any, are assigned a color $c(n)$ between 1 and K, according to a deterministic rule.

4. All nodes n with color $c(n) \neq k$ are just discarded. The remaining nodes are processed (in any order and possibly in a nondeterministic way) till completion.

CP application

- **Constraint Programming** implementation within Gecode (open source)
- ***NODE_PAUSE(n) == true*** if the **estimated difficulty** of node *n* (variable domain volume) is Θ times smaller than that at the root node
- On-the-fly **automatic tuning** of threshold Θ (same rule for all instances)
- After sampling, paused nodes are first sorted by increasing estimated difficulty, and then colors are assigned in **round-robin**
- Results on feasibility instances

instance	time (s) $K = 1$	speedup $K = 4$	$K = 16$	$K = 64$
golomb_12	41.5	3.84	14.31	41.50
golomb_13	1195.8	4.00	15.67	57.49
golomb_14	19051.9	3.97	15.71	61.34
partition_16	30.0	3.75	13.64	46.15
partition_18	354.8	3.90	14.78	54.58
partition_20	4116.4	3.86	15.64	59.40
ortholatin_5	29.3	3.89	13.95	36.63
sports_10	98.7	3.91	14.51	44.86
hamming_7_4_10	32.3	3.85	14.04	40.38
hamming_7_3_6	2402.4	3.91	15.44	59.76

MIP application (B&Cut ATSP)

Sequential code to parallelize: Branch-and-cut FORTRAN code of about 10,000 lines from

- M. Fischetti, P. Toth, “A Polyhedral Approach to the Asymmetric Traveling Salesman Problem” Management Science 43, 11, 1520-1536, 1997.
- M. Fischetti, A. Lodi, P. Toth, “Exact Methods for the Asymmetric Traveling Salesman Problem”, in The Traveling Salesman Problem and its Variations, G. Gutin and A. Punnen ed.s, Kluwer, 169-206, 2002.

Main Features

- LP solver: CPLEX 12.5.1
- Cuts: SEC, SD, DK, RANK (and pool) separated along the tree
- Dynamic (Lagrangian) pricing of var.s
- Variable fixing
- Primal heuristics
- Etc.

Results with 4 and 8 workers
(on a quadcore hyperthreading CPU)

- Random instances taking 1,000 to 4,000 sec.s in sequential mode
- **Paused-node** version (with incumbent written on file)
- **11+46 new lines of code** added to the original source code
- Average speedup of **3.11** (geom.mean **3.09**) with **4 workers**
- Average speedup of **4.38** (geom.mean **4.31**) with **8 workers**

mywork	nwork	opt	elapsed (s)
1	4	12574	870
2	4	12574	871
3	4	12574	870
4	4	12574	870
1	1	12574	2374

mywork	nwork	opt	elapsed (s)
1	8	12574	547
2	8	12574	519
3	8	12574	501
4	8	12574	510
5	8	12574	540
6	8	12574	501
7	8	12574	501
8	8	12574	501
1	1	12574	2374

MIP application (CPLEX)

We performed the following experiments

1. We implemented **SelfSplit** in its paused-node version using CPLEX callbacks.

2. We selected the instances from **MIPLIB 2010** on which CPLEX consistently needs a large n. of nodes, even when the incumbent is given on input, and still can be solved within 10,000 sec.s (single-thread default). **This produced a testbed of 32 instances**.

3. All experiments have been performed in **single thread**, by giving the incumbent on input and disabling all heuristics → approximation of a production implementation involving **some limited amount of communication** in which the incumbent is shared among workers.

MIP application (CPLEX)

Experiment n. 1

We compared CPLEX default (with empty callbacks) with **SelfSplit_1**, i.e. **SelfSplit with input pair (1,1),** using 5 random seeds. The slowdown incurred was just 10-20%, hence Self_Split_1 is comparable with CPLEX on our testbed

Experiment n. 2

We considered the availability of **16 single-thread machines** and compared two ways to exploit them without communication:

(a) running **Rand_16**, i.e. SelfSplit_1 with 16 random seeds and taking the best run for each instance **(concurrent mode)**

(b) running **SelfSplit_16,** i.e. SelfSplit with input pairs (1,16), (2,16),…,(16,16)

MIP application (CPLEX)

	SelfSplit_1		SelfSplit_16			Rand_16		
	time	opt	Time	opt	speedup	Time	opt	speedup
beasleyC3	10,000.01	0	10,000.00	0	1.00	1,601.26	1	6.25
csched007	10,000.01	0	1,445.56	1	6.92	2,166.22	1	4.62
csched010	5,471.81	1	475.61	1	11.50	1,183.64	1	4.62
danoint	2,579.58	1	234.82	1	10.99	1,767.17	1	1.46
enlight16	272.44	1	10.35	1	26.32	154.53	1	1.76
iis-bupa-cov	10,000.01	0	1,762.88	1	5.67	10,000.01	0	1.00
k16x240	3,526.51	1	365.03	1	9.66	3,526.51	1	1.00
mcsched	4,744.56	1	371.82	1	12.76	3,735.39	1	1.27
mik-250-1-100-1	1,131.14	1	1,543.85	1	0.73	1,129.12	1	1.00
momentum1	8,730.25	1	2,224.69	1	3.92	3,476.15	1	2.51
neos-1426662	5,591.18	1	2,980.37	1	1.88	590.31	1	9.47
neos-1442657	639.15	1	83.95	1	7.61	180.05	1	3.55
neos-1616732	2,792.79	1	549.99	1	5.08	1,410.80	1	1.98
neos-1620770	10,000.01	0	208.04	1	48.07	1,356.77	1	7.37
neos-942830	1,626.58	1	3,662.14	1	0.44	258.65	1	6.29
neos15	5,096.19	1	3,081.10	1	1.65	5,094.56	1	1.00
neos16	10,000.01	0	127.86	1	78.21	2,041.42	1	4.90
neos858960	2,043.96	1	173.69	1	11.77	794.92	1	2.57
newdano	10,000.01	0	1,406.58	1	7.11	9,820.67	1	1.02
nobel-eu-DBE	10,000.01	0	7,577.27	1	1.32	5,641.82	1	1.77
noswot	147.14	1	17.39	1	8.46	23.99	1	6.13
ns1766074	89.45	1	10.30	1	8.68	85.88	1	1.04
ns2081729	6,588.38	1	10,000.00	0	0.66	6,588.38	1	1.00
nu60-pr9	10,000.01	0	3,248.67	1	3.08	5,088.17	1	1.97
pg5_34	9,416.21	1	826.22	1	11.40	9,407.77	1	1.00
pigeon-10	421.67	1	65.16	1	6.47	395.49	1	1.07
ran14x18	3,163.16	1	236.98	1	13.35	2,438.44	1	1.30
ran14x18-disj-8	1,709.97	1	201.25	1	8.50	1,709.75	1	1.00
reblock166	10,000.02	0	10,000.00	0	1.00	10,000.01	0	1.00
rmine6	10,000.01	0	1,716.65	1	5.83	4,763.89	1	2.10
rococoB10-011000	10,000.01	0	5,092.32	1	1.96	10,000.01	0	1.00
timtab1	1,675.12	1	171.30	1	9.78	868.85	1	1.93
sum		**21**		**29**			**29**	
avg					**10.37**			**2.69**
geomean					**5.29**			**2.01**

Extensions

- SelfSplit can be run with just $K' << K$ workers, with input pairs $(1,K), (2,K), \ldots,(K',K)$ → kind of **multistart heuristic** that guarantees non-overlapping explorations

- It can be used to obtain a quick **estimate of the sequential computing time**, e.g. by running SelfSplit with (1,1000), …(8,1000) and taking

  sampling_time +

  1000 * (average_computing_time – sampling_time)

- Allows for a **pause-and-resume** exploration of the tree (useful e.g. in case of computer failures)

- Applications to **High Performance Computing** and **Cloud Computing?**

Thank you for your attention