Andorra-I

- a realisation of the Basic Andorra Model;
- a system that transparently exploits both and- and or-parallelism;
- and-scheduler (two run queues of goals) and or-scheduler (Bristol).
- *flexible scheduling of and-work and or-work.*

Andorra-I

```
paper(P,D,I,T) :- date(P,D), author(P,A), loc(A,I,D), title(P,T).
paper(P,D,I,T) :- tr(P,I), date(P,D).
paper(xform,1990,bristol,T) :- tr(xform,T).

author(fp,backus).                date(fp,1978).
author(ppro,khayri).              date(ppro,1990).
author(pprom,warren).             date(pprom,1990).
author(ppro,yang).                date(ppro,1991).
author(orsch,beaumont).           date(orsch,1991).
author(ai,costa).                 date(ai,1991).
author(ai,herme).                 date(ai,1990).
author(sasl,turner).              date(sasl,1979).
author(xform,stonebridge).

title(ppro,muse_prolog_system_and_performance).
title(pprom,aurora_parallel_prolog_model).
title(fp,can_programming_be_liberated).
title(ppro,implementation_basic_andorra_model).
title(sasl,a_new_implementation_technique).
title(ai,compile_time_analysis_andorraI).
title(ai,abstract_interpretation_and_applications).
title(orsch,flexible_scheduling_in_aurora).
title(xform,bristol_tech_reports_catalog).

loc(khayri,sics,1990).            journal(fp,cacm).
loc(backus,ibm,1978).             journal(sasl,spe).
loc(yang,bristol,1991).
loc(beaumont,bristol,1991).       tr(db,edinburgh).
loc(costa,bristol,1991).
loc(herme,madrid,1990).
loc(turner,kent,1981).
loc(warren,sri,1982).
loc(warren,bristol,1990).
```

Figure 1: Database Program (2).

Andorra-I

Figure 2: Andorra-tree example

Andorra-I

Andorra-I original design:

- each worker is associated to a physical processor;
- workers are arranged into *teams*: each team is composed of 1 **master** and a number of **slaves**;
- teams cooperate to execute or-work;
- workers in a team cooperate to execute and-work;

Andorra-I:Parallel Implementation

```
Begin
        Get parameters from command line;
        Allocate and initialise shared memory areas;
        Initalise workers;
        Sched_Init and Sched_Setup worker;
        my_id = 0;
        for (i = 1; i < Total_Workers; i++)
             if ( fork() == 0 ) then;
                my_id = i;
                Sched_Init and Sched_Setup worker;
                Initialise my own variables;
                break;
             Endif
        Endfor
        if Is_Master(my_id) then;
           Execute master code;
        else
           Execute slave code;
        Endif
        wait all finish;
End.
```

Figure 3: ANDORRA-I PARALLEL ALGORITHM

Andorra-I

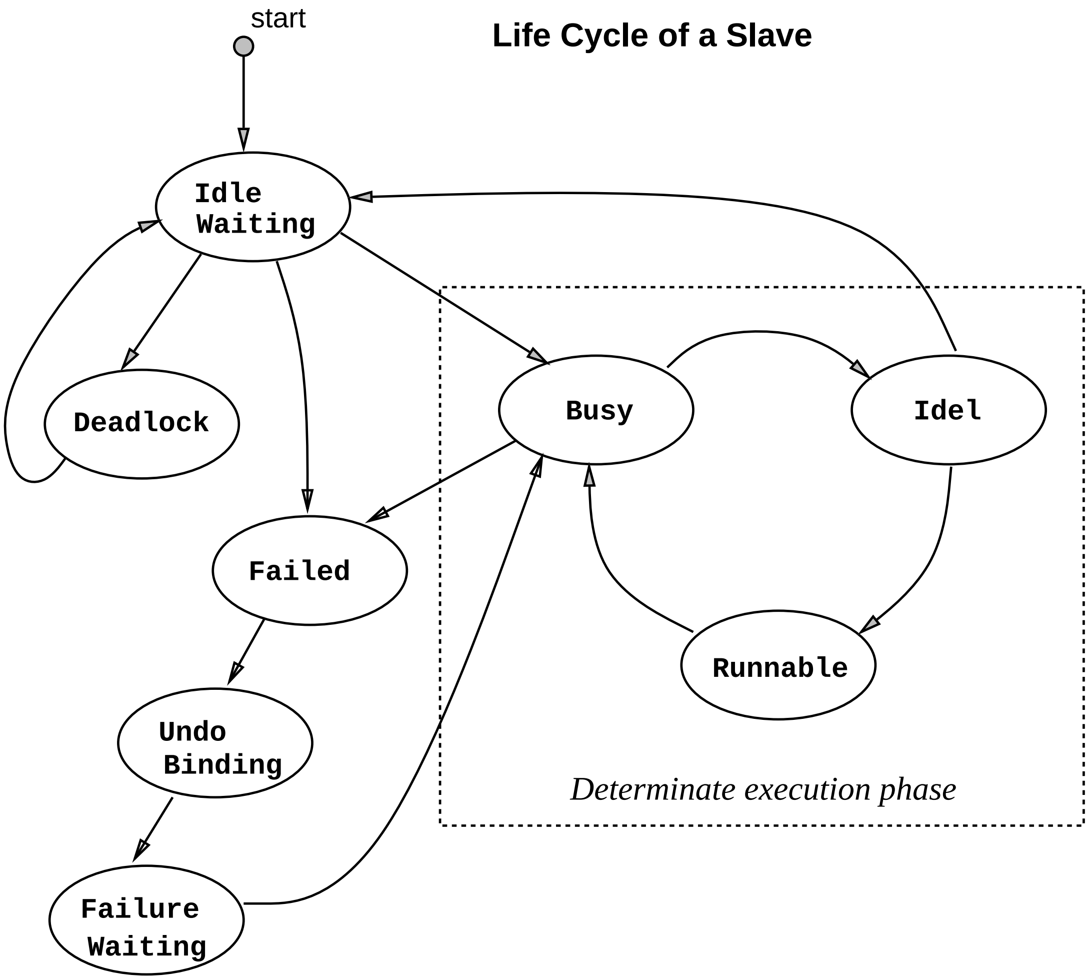


Figure 4: Andorra-I Slave Engine Cycle

Andorra-I

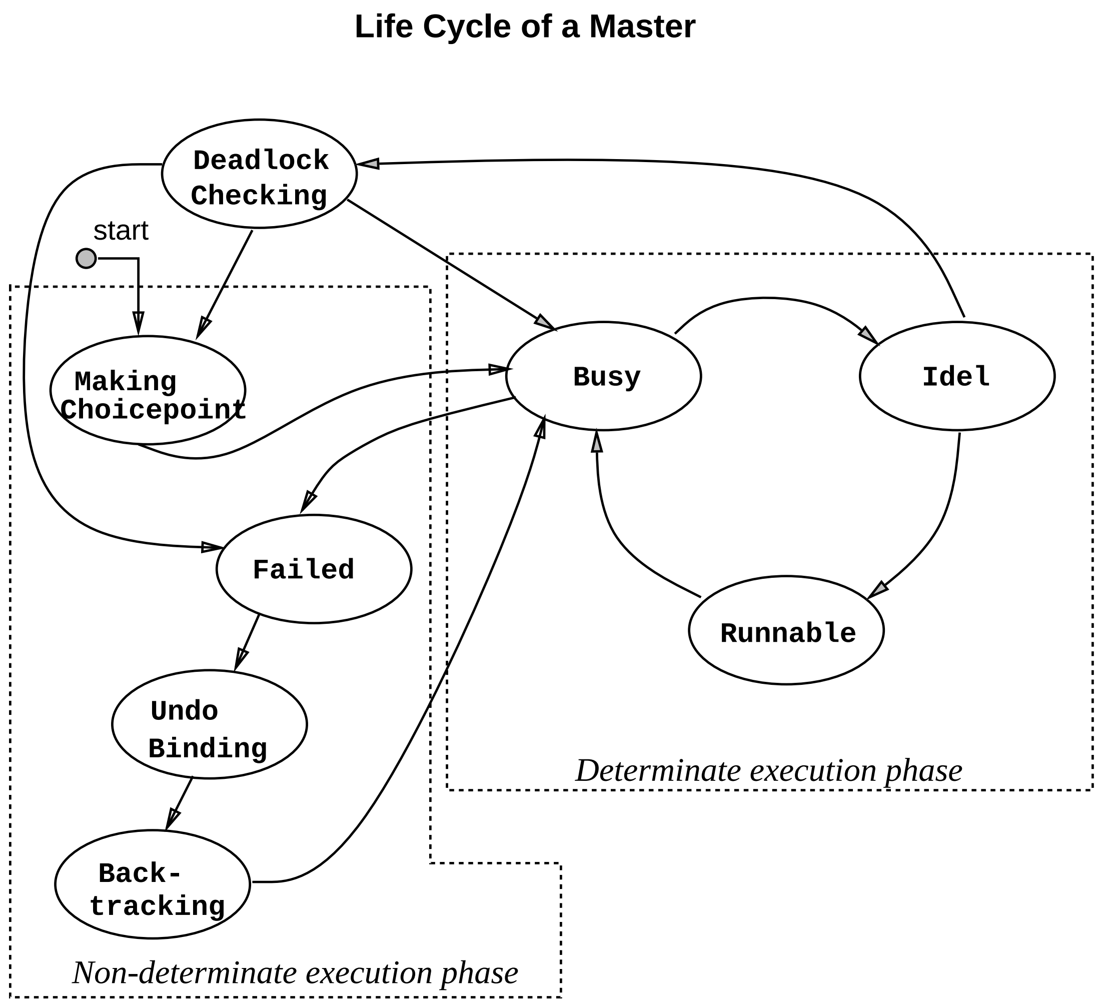


Figure 5: ANDORRA-I MASTER ENGINE CYCLE

Andorra-I

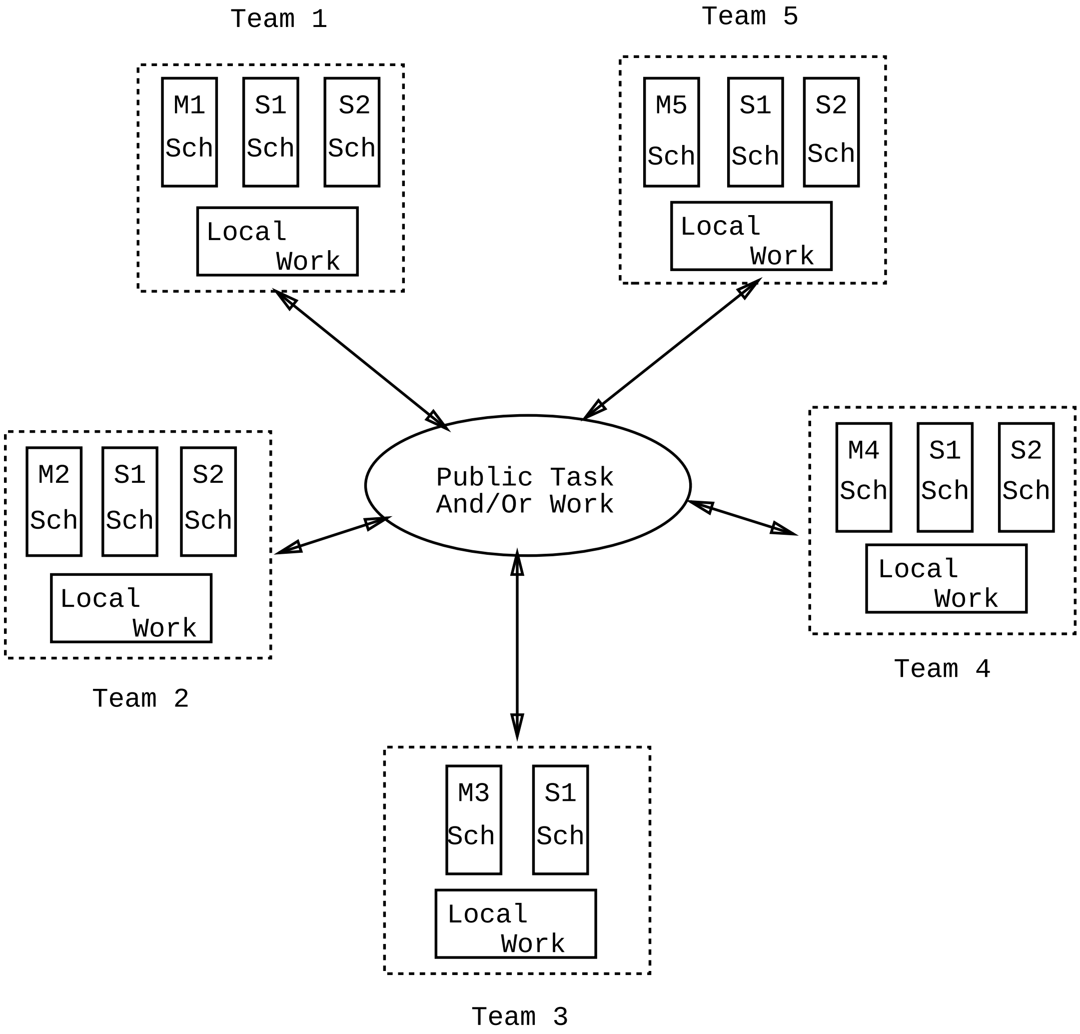


Figure 6: Andorra-I System Architecture

Andorra-I

Figure 7: An Andorra-Tree

Andorra-I

Config	time units
1M4S	26
2M3S	18
3M2S	14
4M1S	14
5M	10

Figure 8: An Andorra irregular tree

Andorra-I: Problems

Problems with Andorra-I original design:

- user needs to manually adjust teams configuration;
- large variations between run times;
- if the parallelism during the computation is not balanced, Andorra-I performs sub-optimally.

Andorra-I new design: *dynamic* reconfiguration of workers into teams.

The Reconfigurer

- *Reconfigurer*:

  → dynamically reconfigures the teams;

  → changes workers:

  from slave to master;

  from master to slave;

  from one team to another.

The Reconfigurer

Figure 9: LIFE CYCLE OF A WORKER

The Reconfigurer

From performance analysis of Andorra-I:

Valid movements for a **slave** worker, in order of cost:

- stay in its original team, move to another team, become a master, be released to the system.

Valid movements for a **master** worker, in order of cost:

- stay as a master, become a slave and move to a new team, be released to the system.

The Reconfigurer

Two solutions:

- *Work-Guided Strategy* (WGS): decisions are taken according to prediction of future work;
- *Efficiency-Guided Strategy* (EGS): decisions are taken according to past computation.

The Work-Guided Strategy

Simple strategy based on sizes of work provided by and-scheduler and or-scheduler:

- size of the run queues;
- number of *live* nodes or alternatives;
- workers move when they become idle;
- or-work compared to and-work when both kinds of work available.
- when moving to another team, a worker searches for the *best* team.

The Work-Guided Strategy

Begin

Get 'w' from and-scheduler;

Get 'o' from or-scheduler;

if $(w = 0) \wedge (o = 0)$ **then**

  return no change;

if $(w = 0) \wedge (o \neq 0)$ **then**

  return change to master;

if $(w \neq 0) \wedge (o = 0)$ **then**

  return no change;

if $(\frac{w}{R} \times D_a < C_a)$ **then** /* $(o \geq C_o) \vee (o < C_o)$ */

  return change to master;

if $(\frac{o}{R} \times D_o \geq C_o) \wedge (\frac{w}{R} \times D_a \geq C_a)$ **then**

  if $(o \geq G_o \times w)$ **then**

    return change to master;

return no change;

End.

Figure 10: Decide change to Master

The Work-Guided Strategy

Begin

Get 'w' from and-scheduler;

Get 'o' from or-scheduler;

if $(w = 0) \wedge (o = 0)$ **then**

  return no change;

if $(w = 0) \wedge (o \neq 0)$ **then**

  return no change;

if $(w \neq 0) \wedge (o = 0)$ **then**

  return change to slave;

if $(\frac{o}{R} \times D_o < C_o)$ **then** /* $(w \geq C_a) \vee (w < C_a)$ */

  return change to slave;

if $(\frac{o}{R} \times D_o \geq C_o) \wedge (\frac{w}{R} \times D_a \geq C_a)$ **then**

  if $(w \geq G_a \times o)$ **then**

    return change to slave;

return no change;

End.

Figure 11: Decide change to Slave

The Work-Guided Strategy

Begin

    For each active team t

        Get w_t from and-scheduler;

        $l_t = \frac{w_t}{R_t} \times D_t$;

    end;

    Get team t^* with maxof(l);

    return t^*;

End.

Figure 12: FIND BEST TEAM

The Work-Guided Strategy

Advantages:

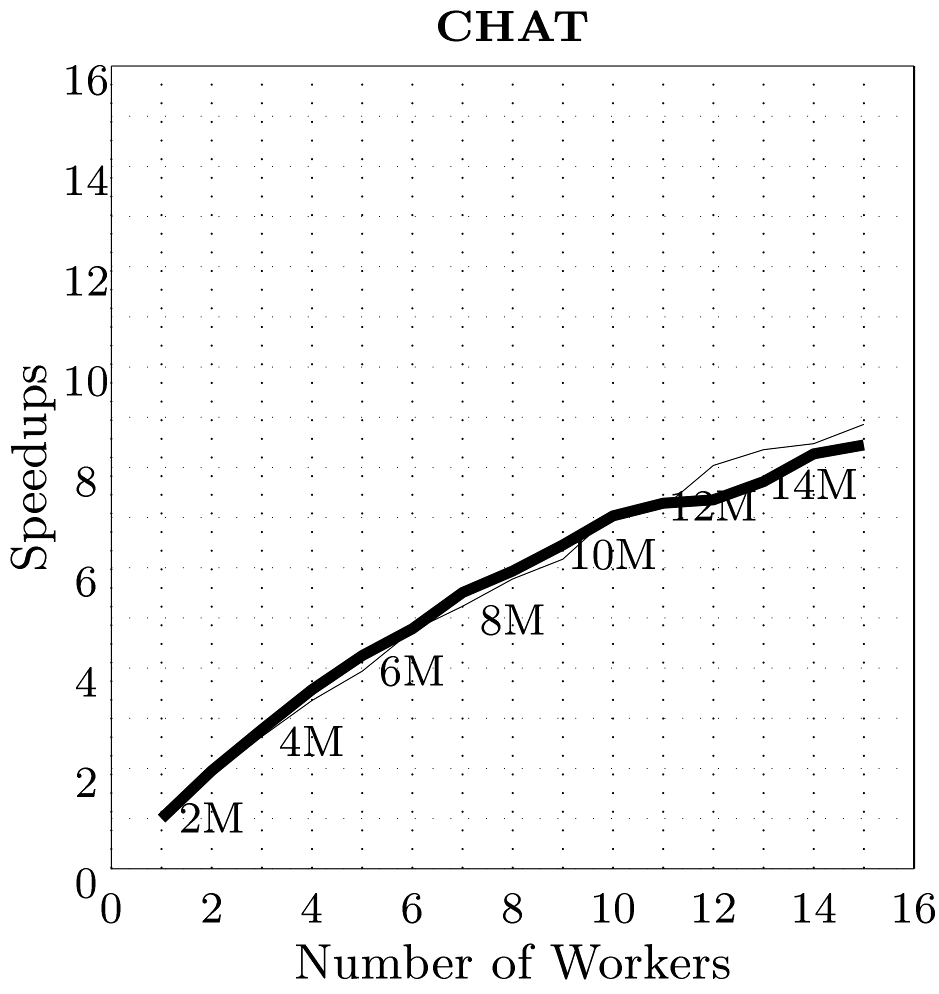
- no manual adjustment is necessary;
- it can produce better results than the fixed config;

Disadvantages:

- Current information based on immediate state of the computation;
- it is not better than fixed for some few computations.

Work-Guided x Best Fixed Teams: 15 procs

Figure 13: Work-guided strategy: only or benchmark



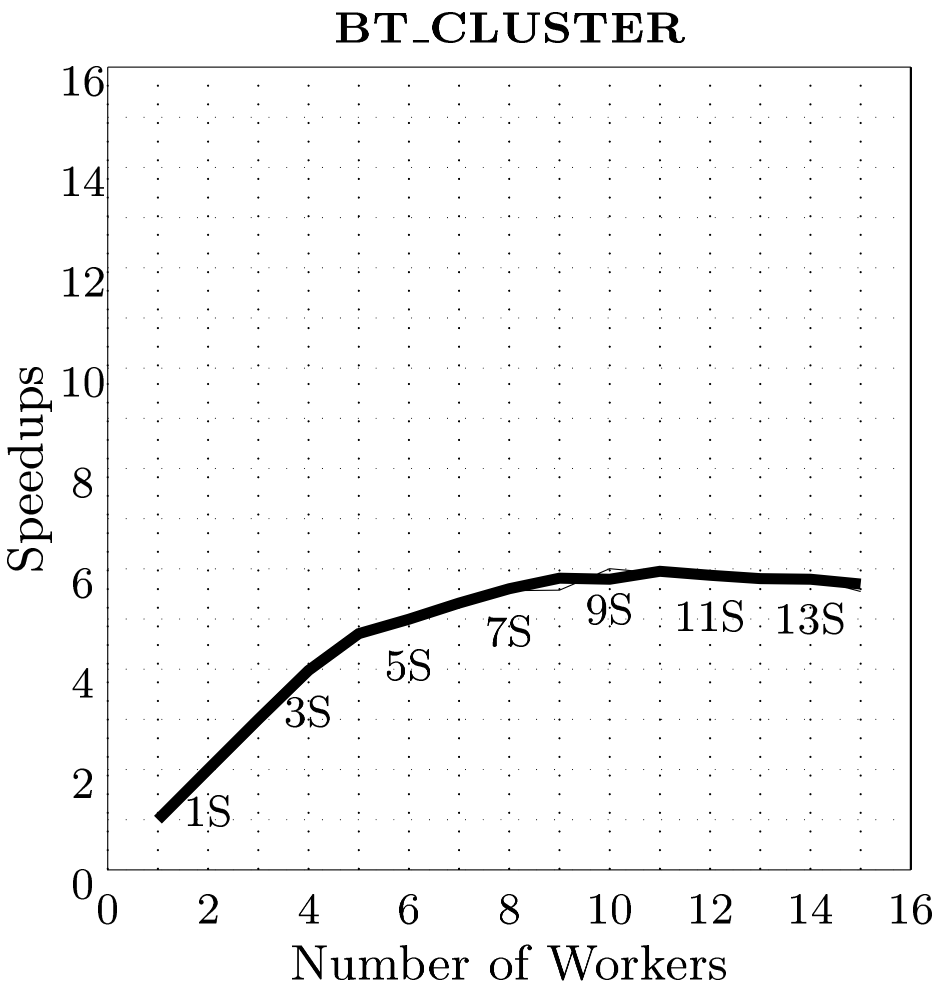
Work-Guided x Best Fixed Teams: 10 procs

Benchmarks	Speedup with weighting, p =					Sp w
	0	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$	1	Rec
ndetdet	2.037	4.673	5.913	7.971	**8.487**	9.831
mixed	0.980	1.920	2.642	**3.819**	3.584	4.004
detndet	1.044	1.622	2.052	**2.387**	2.032	3.157
bqu8	1.527	2.858	3.937	5.487	**7.516**	4.474
chat	0.666	1.578	2.502	3.866	**7.334**	7.034
cypher	1.147	2.164	2.875	4.046	**5.922**	6.116
mutest	0.584	1.386	2.071	3.450	**5.959**	6.025
bqu6	1.499	2.562	2.940	3.132	**3.137**	2.920
flypan2	**6.203**	4.105	3.507	1.964	1.007	6.307
bt_cluster	**6.002**	4.353	3.947	1.995	1.011	5.793
flypan4	**5.079**	3.725	3.305	1.947	1.062	4.961
scanner	5.009	**5.460**	4.992	4.087	3.226	5.414
road_mark	**4.887**	3.452	2.479	1.790	0.965	4.269
bcnet	3.226	**3.763**	3.336	2.767	2.033	3.942
cross11	**2.666**	1.647	1.615	1.669	1.112	2.917
flypan5	2.629	**2.878**	2.506	2.139	1.335	3.382
cross6	**2.529**	2.260	2.095	1.342	0.964	2.260
H.mean	1.656	2.505	**2.787**	2.574	1.836	**4.258**

Table 1: ANDORRA-I IN THE PAST X ANDORRA-I WITH THE RECONFIGURER, AT 10 WORKERS

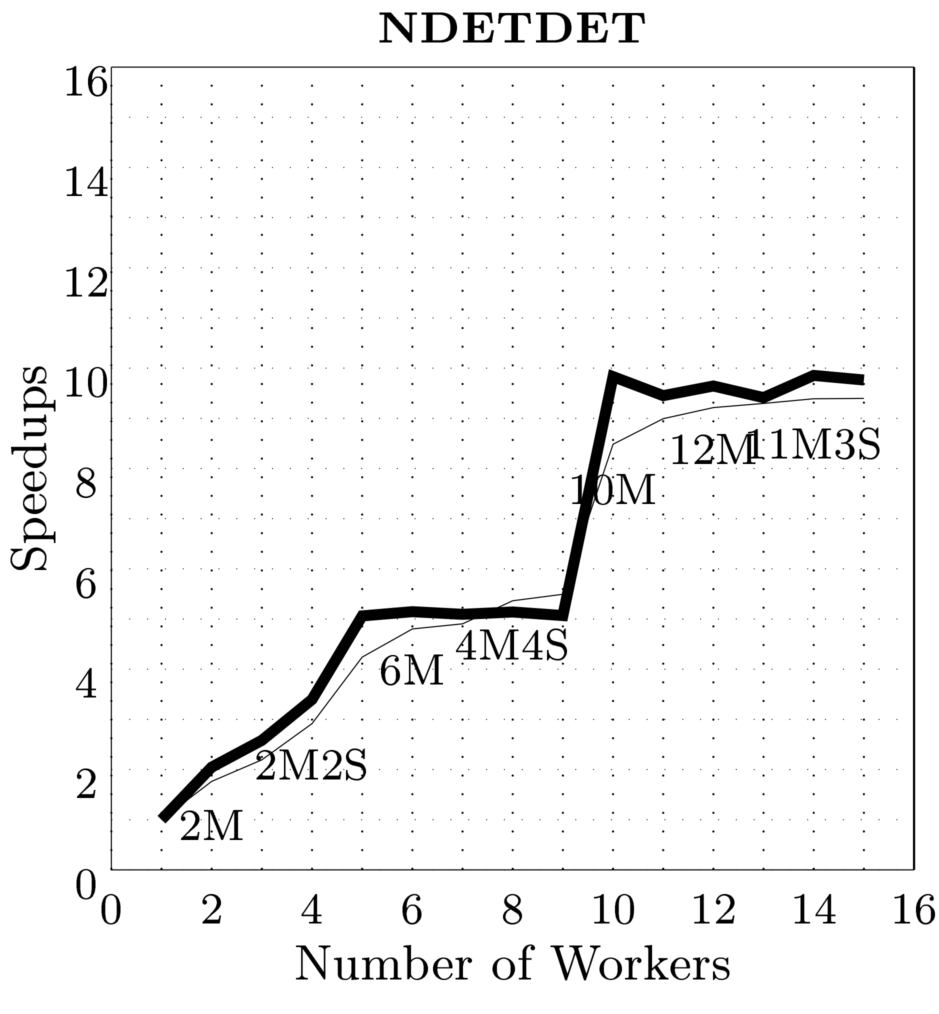
Work-Guided x Best Fixed Teams: 15 procs

Figure 14: WORK-GUIDED STRATEGY: ONLY AND BENCHMARK



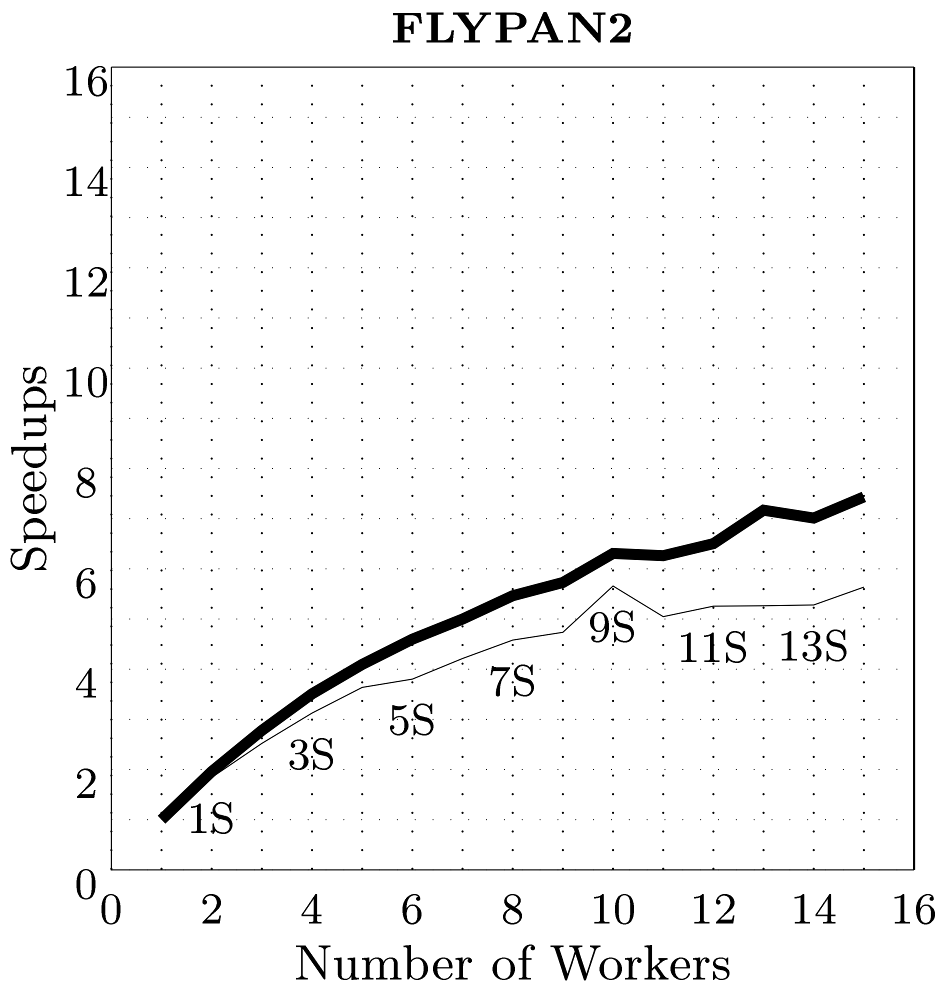
Work-Guided x Best Fixed Teams: 15 procs

Figure 15: Work-guided strategy: high or, low and benchmarks



Work-Guided x Best Fixed Teams: 15 procs

Figure 16: Work-guided strategy: high and, low or benchmarks



The Efficiency-Guided Strategy

- Each worker monitors its own efficiency along the execution time.
- If the efficiency of the worker drops below a threshold, make a valid movement.
- Efficiency can be measured in two ways:
  - number of reductions performed along the time, with respect to the *sequential lips rate.*
  - total **working time** over total execution time.
- Approach taken: **working time**.

The Efficiency-Guided Strategy

Begin

$working\_time$:= Get my working time;
$time$:= Get elapsed time;
$e := \frac{working\_time \times 100}{time}$;
if $e < eff\_threshold$ **then**
  $next\_deployment := (curr\_deployment + 1)$ mod $worker\_deployments$;
  if ($next\_deployment$ = `change teams`) or ($next\_deployment$ = `change to slave`) **then**
    $Find\ best\ team\ t^*$;
    if there is no such team **then**
      return `no change`;
    else
      redeploy slave to team t^*;
      return `change to slave`;
  return $next\_deployment$;
else
  return `no change`;
End.

Figure 17: Master or slave calls reconfigurer

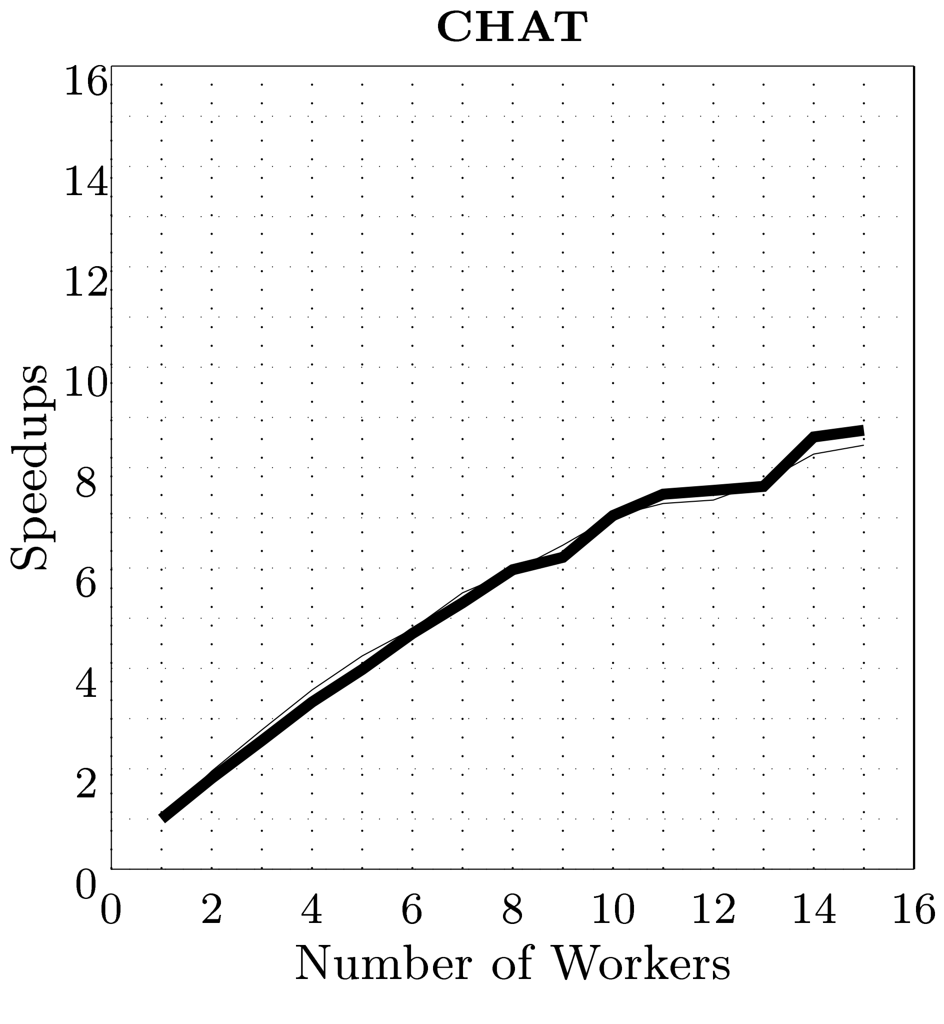
The Efficiency-Guided Strategy

Advantages:

- it can produce better results than the WBS for a class of programs that contain fine grained and-parallelism in or-branches;
- it uses a better heuristic based on the past computation.

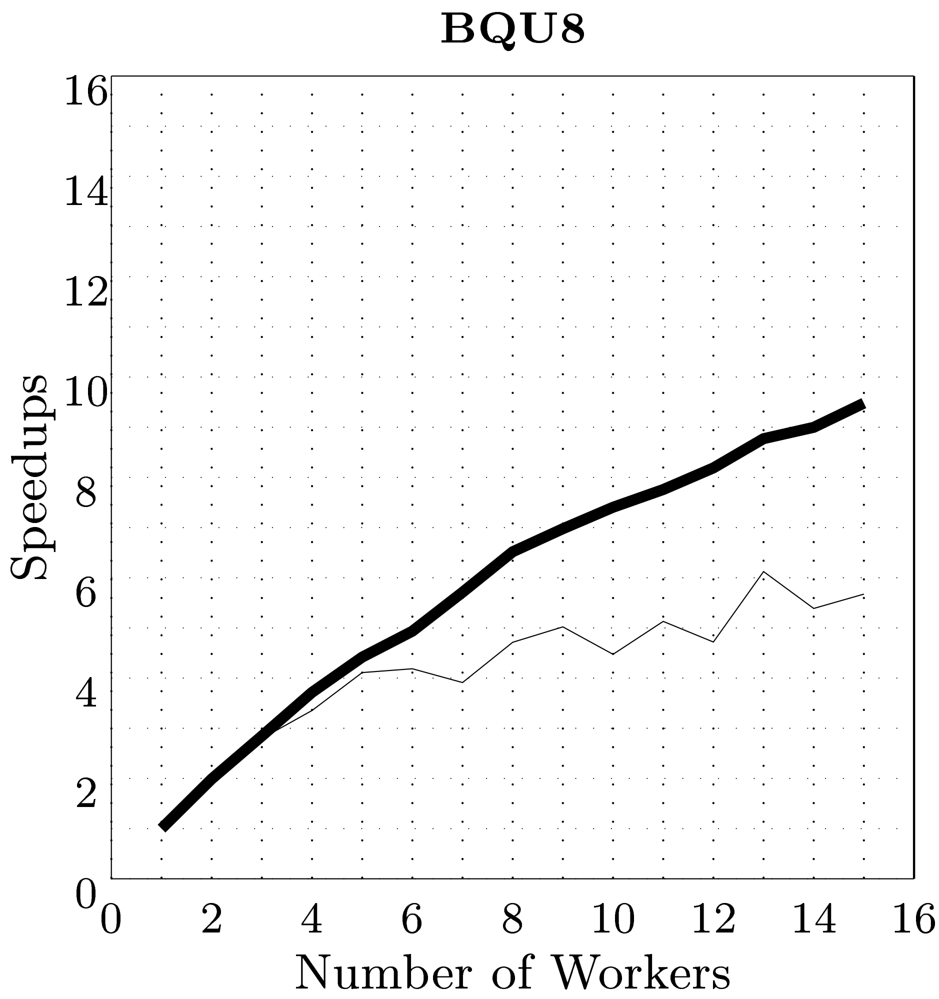
Work-Guided x Efficiency-Guided: 15 procs

Figure 18: Work-guided strategy: only or benchmark



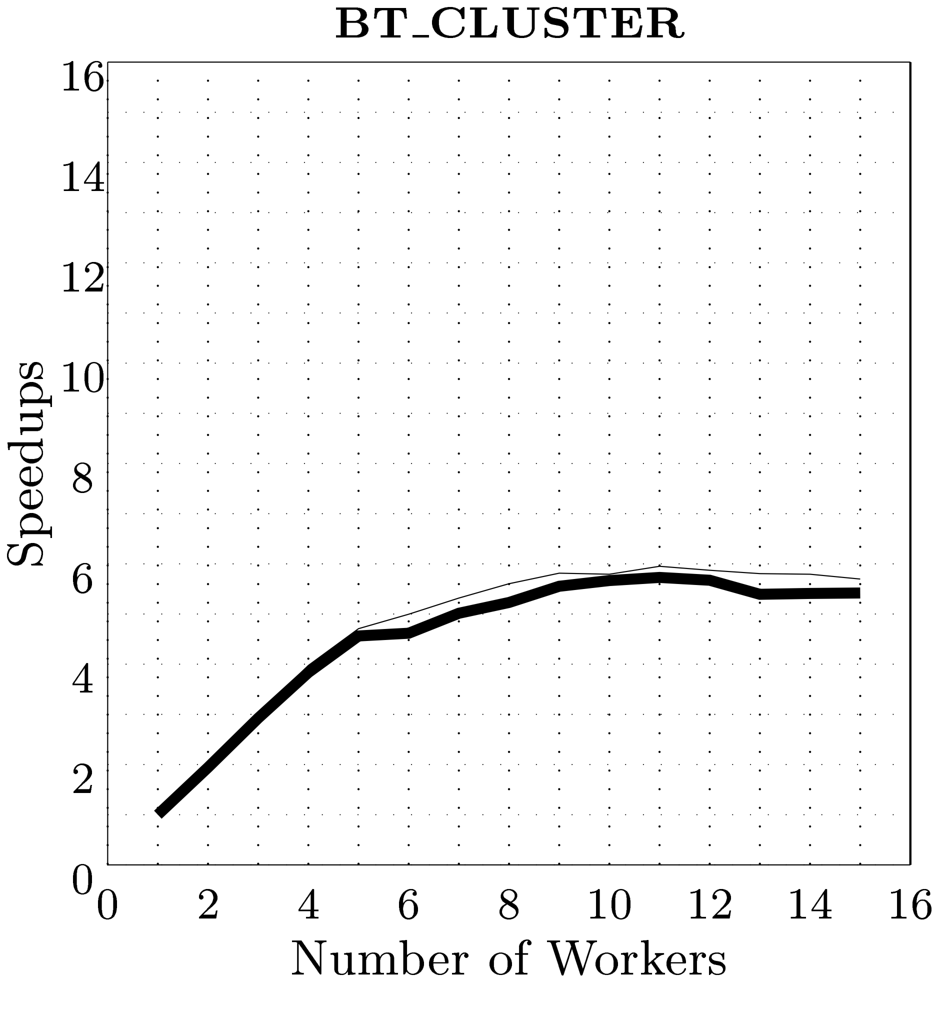
Work-Guided x Efficiency-Guided: 15 procs

Figure 19: Work-guided strategy: high or, low and benchmarks



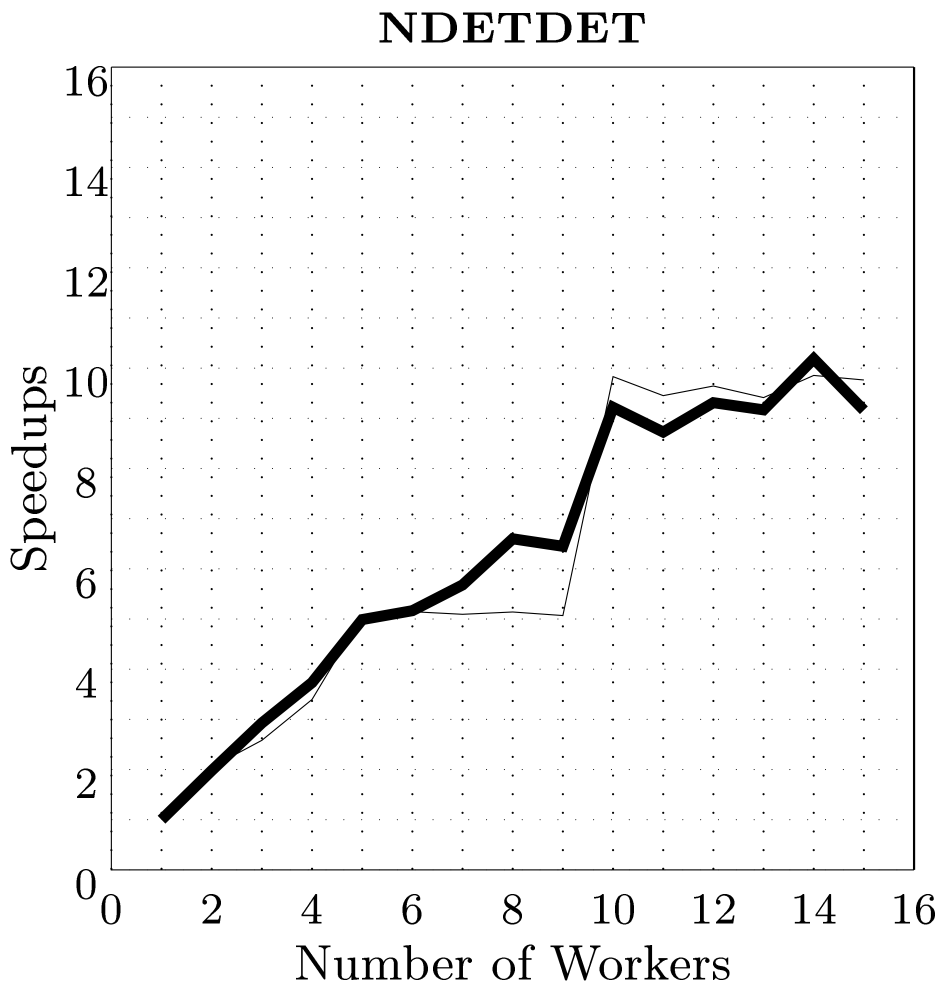
Work-Guided x Efficiency-Guided: 15 procs

Figure 20: WORK-GUIDED STRATEGY: ONLY AND BENCHMARK



Work-Guided x Efficiency-Guided: 15 procs

Figure 21: Work-guided strategy: high or, low and benchmarks



Work-Guided x Efficiency-Guided: 15 procs

Figure 22: Work-guided strategy: high and, low or benchmarks