Vrije Universiteit Brussel

Applied Sciences and Engineering: Computer Science

Profile Profile Software Languages and Software Engineering

Declarative Programming Project : Task-Parallel Scheduler

Nicolas Omer
E-mail : nicoomer@vub.ac.be
Student number : 0516475

August 16, 2015
Academic Year 2014–2015

Contents

1 Introduction

For the project, an offline scheduler for task-parallel computations using SWI-Prolog had to be implemented. To test the scheduler, various problem instances were provided. Each defines the properties of the computation (as a DAG of tasks) and execution environment (e.g. number of processing cores).

2 Optimal Solution Approach

A classic brute force algorithm was used in order to find the optimal solution for the 5 small instances (while taking in account the heterogeneity of the system the tasks dependencies and the communication costs). The scheduler will build every possible valid solution that does not violate the constraints : all cores are present in the solution (even if they do not schedule any task), all tasks must be scheduled, tasks dependencies are respected (valid topological order), there are no duplicate schedules for cores and tasks are executed only once.

To do so, the scheduler will first of all list the available tasks that can be scheduled (the ones that have no dependencies) and will check if it is the first non executed task of its core in the given solution. The available tasks have to be scheduled first because once their execution is done, new tasks will be available for schedule since that their dependent tasks are done.

To be able to keep track of which tasks are already scheduled and which are not, the optimal(TodoTasksList, SolutionsPCWCList, Solution) functor maintain a SolutionsPCWCList that is a list with tuples (SolutionsList, processCostWithCore(TriplesList)). For each Solution in the list, we have a list with processCostWithCore(Task, Core, Cost) to recall the tasks that were scheduled.

This SolutionsPCWCList is modified through a great number of helper function (building partial solution) and it is used to compute the earliest starting time of a task / ending time / execution time of a Solution. The processCostWithCore part is discarded at the end, leaving only the SolutionsList. A big aggregate predicate is used to choose the Solution that has the smallest execution time :

```
aggregate(min(ET, X), (member(X, SolutionsList), isSolution(X),
  execution_time(X, ET)), min(_, Solution)).
```

3 Speedup Factor Approach

Lastly, the speedup ratio (ET / ET1) is computed with the following technique : the execution time is determined for the optimal solution (or the heuristic one), it returns ET. For ET1 however, a list of solution is built where every solution is a single core solution (all tasks are scheduled on one core). The fastest solution is returned and it gives us the ET1 value, the (ET / ET1) division corresponds to the Speedup Factor.

4 Heuristic Solution Approach

On the other hand, for the large instances, listing all possible solutions is not achievable since it would take a tremendous amount of time to do so (the brute force algorithm scales in an exponential factor with the size of the set and the batch_large_homo.pl file has already 16 cores and 320 tasks...). To compute an heuristic solution, the scheduler first starts with a single core solution (defined earlier) and performs random permutations of random tasks among the solution : for instance, it tries to schedule a random task, let us say t3, on a random core, c2 and this permutation gives a better execution time, keep it !

In the algorithm itself, it uses only a Raw set of data (Core, Task) because managing a processCostWithCore was to hard to handle throughout all permutation (especially for one solution). The permutations keep going on until the user-fixed deadline (in seconds) is reached. For instance here is what happens if the user tries to evaluate consultBSHO. (to work with the batch_small_homo.pl file) then asks for heuristicPrint(10) (The permutations will keep going on for 10 seconds) :

- The fastest single core solution is computed :

```
solution([schedule(c1,[]),schedule(c2,[]),
      schedule(c3,[]),schedule(c4,[t1,t2,t3,t4,t5,t6,t7])])
```

- It is the best solution so far (with execution time = 400), try one permutation : for instance, put t1 in the schedule of core c1, resulting in :

```
solution([schedule(c1,[t1]),schedule(c2,[]),
      schedule(c3,[]),schedule(c4,[t2,t3,t4,t5,t6,t7])])
```

- >execution time = 300, better solution : keep it.
- Is there still time left ? Yes - >continue permutations ; No - >Return the best solution so far and compute speedup.

5 Strengths & Weaknesses

The advantages of the implementation :
+ The scheduler manages to find the correct optimal schedules with minimal executions times along with the speedup factors for the small instances. The scheduler also finds heuristic schedules, their speedup factors.
+ The time to find the optimal solution of the small instances is reasonable (40 seconds for the 5 instances combined) and the fact that the search for heuristic solutions has time as a parameter (give better solution the longer we left him do his random permutations).
+ User friendly, the scheduler was made to be easy to use and understand (with meaningful names for the variables and functions for instance).

The disadvantages of the implementation :
- While searching for heuristic solutions, if the DeadLine is too big, it sometimes causes a "Out of local stack" maybe because cuts are not being enough / correctly used.
- The code tends to be a bit confusing, it is difficult to regroup a clear functionality inside a single function. The functions calls are going a bit everywhere, making it difficult to trace the flow of execution.
- Lack of reusability, I tend to create a new function as soon as I need something even a bit different from something I already have.

6 Reported Experimental Results

6.1 Optimal Reported Results

```
?- mainOptimal.
Finding all optimal solutions...

Consulting batch_small_homo.pl...
% batch_small_homo.pl compiled 0.00 sec, -637 clauses

Computing optimal solution...
Computing Speedup factor...

Solution found :
-----------------
```

```
Schedule found for core c1 : t1.
Schedule found for core c2 : t4--t5.
Schedule found for core c3 : t3--t6.
Schedule found for core c4 : t2--t7.

Schedule Cost Time : 100
With Speedup : 4

Optimal solution found in : 7.539 sec

Consulting batch_small_hetero.pl...
% batch_small_hetero.pl compiled 0.03 sec, 15 clauses

Computing optimal solution...
Computing Speedup factor...

Solution found :
-----------------

Schedule found for core c1 : t4--t6.
Schedule found for core c2 : t1.
Schedule found for core c3 : t2--t7.
Schedule found for core c4 : t3--t5.

Schedule Cost Time : 100
With Speedup : 3.6

Optimal solution found in : 7.65 sec

Consulting fib_small_nc.pl...
% fib_small_nc.pl compiled 0.07 sec, -12 clauses

Computing optimal solution...
Computing Speedup factor...

Solution found :
```

```
-----------------

Schedule found for core c1 : []
Schedule found for core c2 : []
Schedule found for core c3 : t2--t4.
Schedule found for core c4 : t1--t3--t5--t6--t7.

Schedule Cost Time : 50
With Speedup : 1.4

Optimal solution found in : 10.087 sec

Consulting fib_small_uc.pl...
% fib_small_uc.pl compiled 0.10 sec, 2 clauses

Computing optimal solution...
Computing Speedup factor...

Solution found :
-----------------

Schedule found for core c1 : []
Schedule found for core c2 : []
Schedule found for core c3 : t2.
Schedule found for core c4 : t1--t3--t4--t5--t6--t7.

Schedule Cost Time : 60
With Speedup : 1.1666666666666667

Optimal solution found in : 10.145 sec

Consulting sor_small.pl...
% sor_small.pl compiled 0.14 sec, 29 clauses

Computing optimal solution...
Computing Speedup factor...
```

```
Solution found :
-----------------

Schedule found for core c1 : t1--t2.
Schedule found for core c2 : t3--t6.
Schedule found for core c3 : t4.
Schedule found for core c4 : t5.

Schedule Cost Time : 174
With Speedup : 1.4252873563218391

Optimal solution found in : 2.013 sec
Find all optimal solutions took : 37.785 sec
```

6.2 Heuristic Reported Results

```
?- mainHeuristic(4). % Spend 4 seconds by instance to improve a Solution
Find all heuristic solutions...

Consulting batch_large_homo.pl...
% batch_large_homo.pl compiled 0.02 sec, 609 clauses

Computing heuristic solution...
Starting random permutations...

DeadLine Reached !
Computing Speedup factor...

Schedule Cost Time : 1301
With Speedup : 12.37663335895465

Heuristic solution found in : 5.713 sec

Consulting batch_large_hetero.pl...
% batch_large_hetero.pl compiled 0.13 sec, 4,801 clauses
```

```
Computing heuristic solution...
Starting random permutations...

DeadLine Reached !
Computing Speedup factor...

Schedule Cost Time : 1332
With Speedup : 11.47897897897898

Heuristic solution found in : 5.781 sec

Consulting fib_large_nc.pl...
% fib_large_nc.pl compiled 0.02 sec, -4,830 clauses

Computing heuristic solution...
Starting random permutations...

DeadLine Reached !
Computing Speedup factor...

Schedule Cost Time : 1250
With Speedup : 2.12

Heuristic solution found in : 6.209 sec

Consulting fib_large_uc.pl...
% fib_large_uc.pl compiled 0.01 sec, 2 clauses

Computing heuristic solution...
Starting random permutations...

DeadLine Reached !
Computing Speedup factor...

Schedule Cost Time : 1330
```

```
With Speedup : 1,99248120300752

Heuristic solution found in : 6.227 sec

Consulting sor_large.pl...
% sor_large.pl compiled 0.04 sec, 1,040 clauses

Computing heuristic solution...
Starting random permutations...

DeadLine Reached !
Computing Speedup factor...

Schedule Cost Time : 7170
With Speedup : 1.1827057182705718

Heuristic solution found in : 6.285 sec
Find all heuristic solutions took : 30.436 sec
```