Word Search

The problem

Your task is to produce a solution to a word search problem; this is the sort of thing printed in crossword magazines and so on. You're given a grid of letters and have to find a word which appears horizontally or vertically in the grid. The grid consists only of the 26 letters in the range a…z and the words are not dictionary words, just random collections of up to 24 characters with an average length of 9 characters. There's nothing special in those values, they're just based on the corresponding values from `/usr/share/dict/words` on my computer.

In our case we're only interested in words which appear vertically from top down and horizontally left to right. You are to flesh out the following implementation:

```
class WordSearch(object):
    def __init__(self, grid):
       pass

    def is_present(self, word):
       return True
```

In the `__init__` method, `grid` is a string representing the entire word grid, with each row concatenated; there is no intervening separator, instead you will have to rely on the fixed row length `ROW_LENGTH` to determine the end of a row.

The `is_present` method returns `True` if `word` is present in the grid according to our rules.

Your class will be used as follows

```
ws = WordSearch(grid)

for word in words_to_find:
    if ws.is_present(word):
       print "found {}".format(word)
```

Now the twist: the grid you will be given is a square grid with 10,000 (10^4) letters in each direction, i.e `ROW_LENGTH = 10000`. `words_to_find` is an array of 1,000,000 (10^6) words. And we're looking for a solution which is optimised for run time efficiency. The sizes are not chosen for any specific magic reason, just to indicate that this is a "big" problem.

We're not looking for micro-optimisations in the code: if that were the case dispensing with Python might be a better idea! Instead look at the algorithm side of things.

As a bonus question: how would you go about taking advantage of a multicore system?