Test results for parallel in Graphite

lifeng

May 17, 2009

1 How I make test

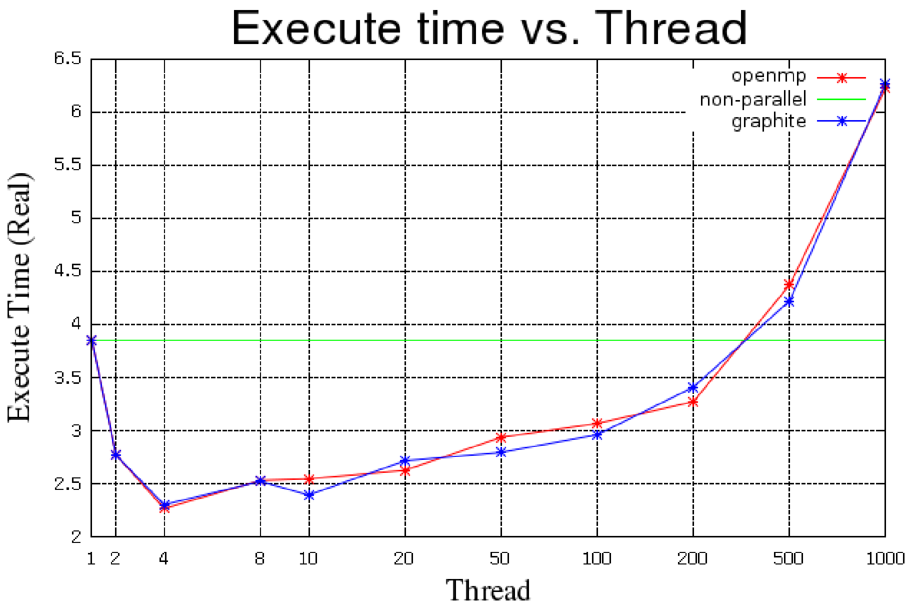
1. All the tests are made in gcc16 which is a 8-core machine when it's CPU usage is 0%. gcc16: 580G 2x4x2.2 GHz Opteron 8354 "Barcelona B3" / 16 GB RAM.
2. Inorder to make sure Graphite make correct openmp code, I also tested with identical openmp c code. So there is two parallel code tests and one non-parallel code test:
   - Runtime of openmp directive inserted c code.
   - Runtime of Graphite autoparallel c code.
   - Runtime of non-parallel c code.
3. The options:
   - `-O2 -fopenmp for openmp directive inserted c code.`
   - `-O2 -fgraphite-force-parallel -ftree-parallelize-loops=$THREAD for Graphite`
   - `-O2 for ordinary non-parallel code.`

2 Test results

2.1 One nested loop tests

This is the same code as in testsuites/.../force-parallel.c. For a accurate time test, I add a k-loop to make it run 1000 times. The #pragma part will be added when testing with openmp run-time, and will removed when testing with Graphite and normal.

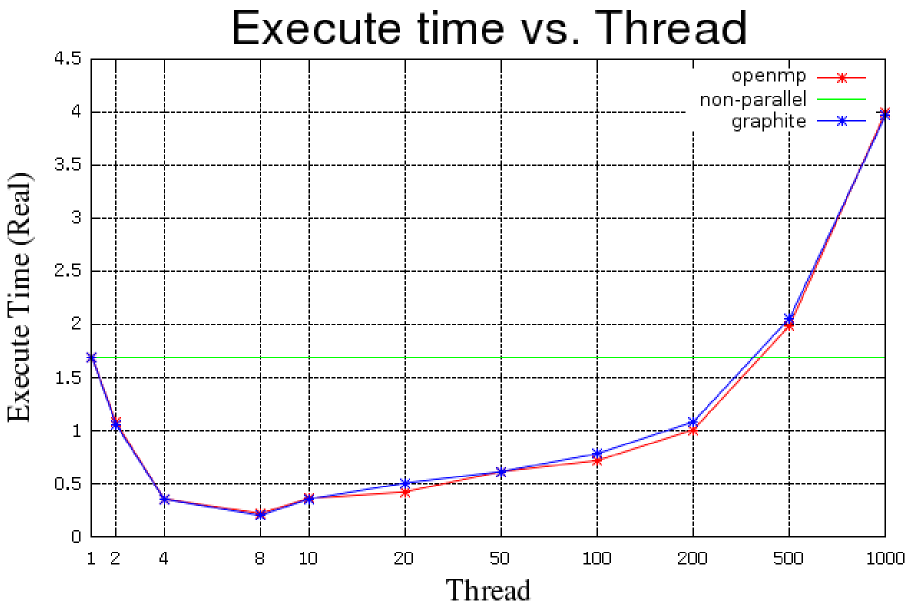
```
#include <stdlib.h>
void parloop (int N)
{
  int i, k;
  int x[1000000];
  for (k = 0; k < 1000; k++){
#pragma omp parallel for shared(N,x) private(i) num_threads(2000)
    for (i = 0; i < N; i++)
      x[i] = 2*(i + 3);

    for (i = 0; i < N; i++)
      {
        if (x[i] != 2*(i + 3)){
          abort();
        }
      }
  }
}

int main(void)
{
  parloop(1000000);

  return 0;
}
```



We could see that Graphite make the same thing as the openmp code. We could see that when parallel, the run-time will decrease firstly as threads increases. But if there is too many threads (overloaded), the run-time will finally increase.

The run-time when parallel with 4-threads did not decrease to 1/4 as we can see in this picture due to a low parallel fraction with Amdahl's Law[1] (Notice we don't have all the code parallel).

I also tested with the non-parallel part(abort() part) removed. This is the results: The fraction is nearly 1, so it may looks reasonable when threading number is small. When threading number is high, the overload caused by parallel increased, so we cann't use Amdahl's Law anymore.



2.2 Two nested loops test.

This is the same code as in force-parallel-2.c. For an acurate test, I add a k-loop.

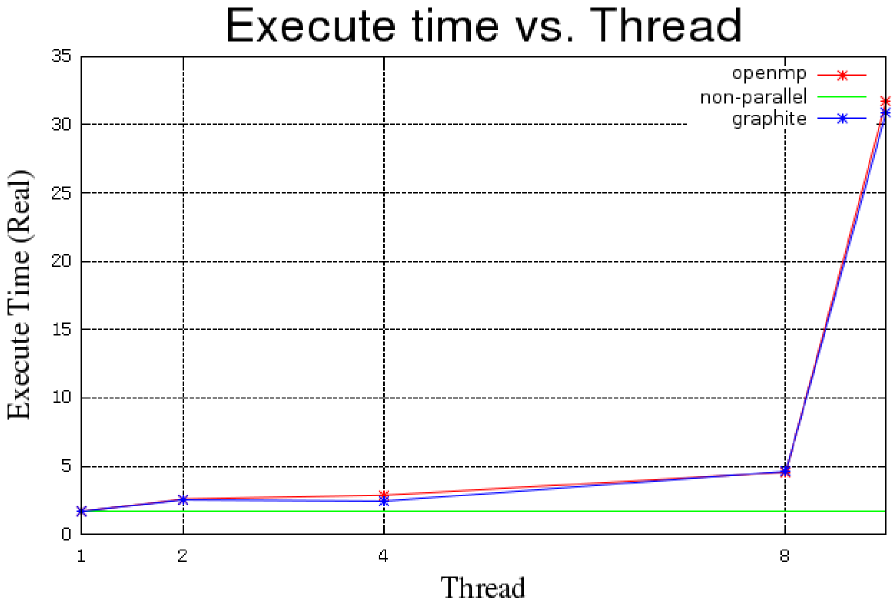
```
#include <stdlib.h>
#define N 1000

int x[N][N];

int main(void)
{

  int i, j, k;
  for (k = 0; k < 1000; k++)
    {
```

[1] http://en.wikipedia.org/wiki/Amdahl%27s_Law

```
      for (i = 0; i < N; i++)
#pragma omp parallel for private(j) num_threads(2)
        for (j = 0; j < N; j++)
          x[i][j] = i + j + 3;
    }
  for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
      if (x[i][j] != i + j + 3)
        abort ();
  return 0;
}
```

2.2.1 It get worse with innermost parallel independent of num_threads.

The test results: We could see from this picture, with the innermost loop parallelized, the run time after parallelization got worse. With a k-loop we added and a i-loop, the barriers will run 1000*1000 times. This is mainly caused by the small number of directives to be ran which is limited by memory.



2.2.2 What if we get outer loop parallel.

As autopar did not deal with outer loop, the test is only tested with openmp. It show a good result when outerloop got parallelized.

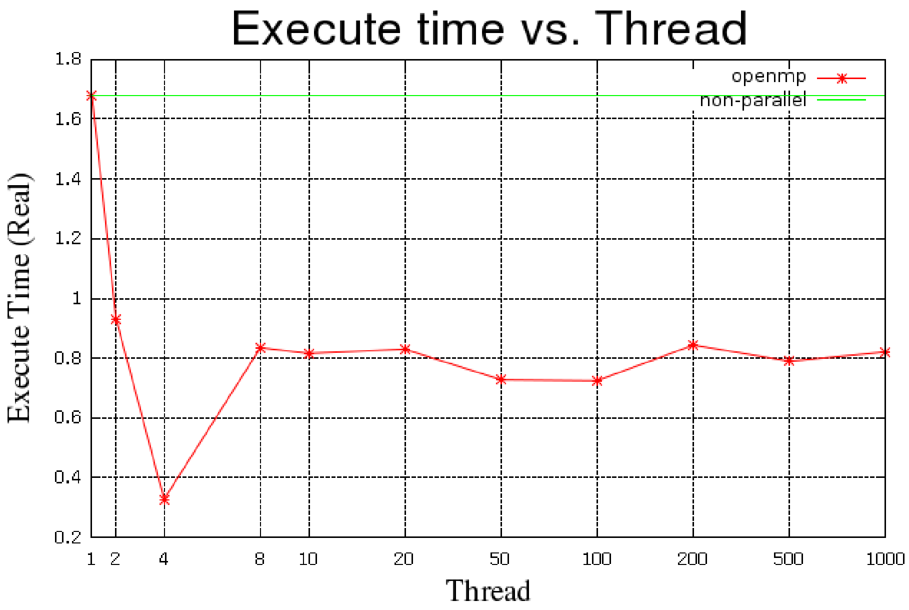
The relative part of code:

```
#pragma omp parallel for private(k) num_threads(2)
```

```
for (k = 0; k < 1000; k++)
  {
    for (i = 0; i < N; i++)
```

Where we parallel with k.

The result: We could see that even threads overload increases, it could also play a good performance.



3 Conclusion

This is only some primary tests with parallelization's performance. The test didn't cover false sharing[2]. The primary results shows that the performance depends. And parallelize the innermost, is not a good idea under some situation.

[2] http://en.wikipedia.org/wiki/False_sharing