Report of Gaussian Elimination using OpenMP

- **Algorithm Description & Correctness Argument**

The code below is the Gaussian Elimination using OpenMP, there are three loops, the inner two loops are data depended on the norm loop, we cannot parallel it, thus I parallel the row loop.

I use 'omp parallel for' to parallel it, every thread should has its own row, col and multiplier, so I make row, col and multiplier private. Since every element in the thread is private, there should be no data dependence.

And since the row loop has less fork and joins, we can leave alone the synchronization, there is no such problem.

To change the performance, I set different schedule and different number of threads.

```
void gauss() {

  int norm, row, col;  /* Normalization row, and zeroing
                          * element row and col */
  float multiplier;

  /* Gaussian elimination */
  /* Since the inner two loops depend on the norm loop, we cann't parallel the norm loop
   * because of the data dependence, I parallel the row loop using omp parallel for,
   * and since each parallel thread has its own row col and multiplier, these values should be
private.
   * I tried several ways to schedule the parallel, static, dynamic and guided, it seems that
guided can
   * achieve the best performance.
   */

  for (norm = 0; norm < N - 1; norm++) {
    #pragma omp parallel for private(row, col, multiplier) schedule(guided) num_threads(4)
      for (row = norm+1; row < N; row++) {
        multiplier = A[row][norm] / A[norm][norm];
        for (col = norm; col < N; col++) {
            A[row][col] -= A[norm][col] * multiplier;
        }
        B[row] -= B[norm] * multiplier;
      }
  }
}
```

- **Different Version's Performance & Performance Analyze**

My laptop has 4 cores, and according to tests, the optimal number of threads is equal to the number of cores.

If the number of threads is less than 4, definitely it can't reach the best performance since there are redundant cores .

And if the number of threads is growing, according to the test, the system CPU time for parent will increase, the performance will degrade rather than improve.

What's more, I also added three different schedules, static, dynamic and guided. According to the test, dynamic has the best performance, however, dynamic is not steady, if the sizes of chunks are allocated unbalanced, the performance will be poor. Schedule guided is the same, although it is steadier than dynamic, sometimes it will still get an abnormal and poor performance.

As for static, it has the steadiest performance. Although the performance may not be the best, it is acceptable and does not differ from other two schedule too much.

So I choose schedule static at last.

And different chunk sizes don't affect the performance much, I find that when the chunk size is around 100, the average performance is the best. I set the chunk size to be 128.

Here is the test result of different versions for 2000*2000 matrix.

Original serial program running time: 8483.16ms

Running time:

	Dynamic	Guided	Static
2 threads	4849.15ms	4907.12ms	4715.58ms
4 threads	4387.18ms	4560.54ms	4618.01ms
8 threads	4434.82ms	4471.45ms	4748.91ms
100 threads	5457.64ms	5190.67ms	6423.17ms

Speedup:

	Dynamic	Guided	Static
2 threads	1.75	1.73	1.80
4 threads	1.93	1.86	1.84
8 threads	1.91	1.90	1.79
100 threads	1.55	1.63	1.32

With chunk size 128:

	Dynamic	Guided	Static
4 threads	4465.7ms	4510.36ms	4481.13ms
4 threads/fault	10507.4ms	11292.6ms	

Speedup:

	Dynamic	Guided	Static
4 threads	1.90	1.88	1.89
4 threads/fault	0.81	0.75	