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 static
can
   * achieve the acceptable and steady performance.
   */

  for (norm = 0; norm < N - 1; norm++) {
    #pragma omp parallel for private(row, col, multiplier) schedule(static)
      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**

According to tests, the optimal number of threads is equal to the number of cores. We can use default setting, and OpenMP will automatically get the number of cores and set number of threads to number of cores.

If the number of threads is less than the number of cores, 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 default, the average static performance is the best. I set the chunk size to be default.

Here is the test result of different versions for 2000*2000 matrix on my own computer.

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
4 threads/fault	10507.4ms	11292.6ms	
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
4 threads/fault	0.81	0.75	
8 threads	1.91	1.90	1.79
100 threads	1.55	1.63	1.32

4 threads and change chunk size:

	Dynamic	Guided	Static
Chunk size: default	4403.97ms	4287.99ms	4399.51ms
Chunk size: 64	4477.44ms	4447.08ms	4650.89ms
Chunk size: 128	4540.4ms	4578.86ms	4702.92ms

Speedup:

	Dynamic	Guided	Static
Chunk size: default	1.93	1.98	1.92
Chunk size: 64	1.89	1.91	1.82
Chunk size: 128	1.87	1.85	1.80

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

Original serial program running time: 31011.3ms

Running time(default chunk size and number of threads changes):

	Dynamic	Guided	Static
2 threads	17505.3ms	17484.6ms	16919ms
4 threads	17520.2ms	17763.6ms	18617.1ms
4 threads/fault	35443.3ms	36520.8ms	
8 threads	17599.7ms	17638.4ms	17548.7ms
100 threads	18503ms	18630ms	19872.9ms

Speedup:

	Dynamic	Guided	Static
2 threads	1.77	1.78	1.83
4 threads	1.77	1.75	1.67
4 threads/fault	0.87	0.85	
8 threads	1.76	1.76	1.77
100 threads	1.68	1.66	1.56

Running time(2 threads and chunk size changes):

	Dynamic	Guided	Static
Chunk size: default	17505.3ms	17484.6ms	16919ms
Chunk size: 64	17843.8ms	17473.6ms	17468.9ms
Chunk size: 128	18020.8ms	17432.6ms	17397.9ms

Speedup:

	Dynamic	Guided	Static
Chunk size: default	1.77	1.78	1.83
Chunk size: 64	1.74	1.78	1.78
Chunk size: 128	1.73	1.78	1.79