Question 1 code:

```
#include "stdlib.h"
#include "stdio.h"
#include "omp.h"

static long numSteps = 100000;
double step;

int main(int argc, char * argv[]){
  argc--; argv++;
  int numthreads = 4;
  if(argc > 0) numthreads = atoi(argv[0]);

  int i;
  double pi, sum = 0.0;
  double partialsum[numSteps];

  step = 1.0/(double)numSteps;

  omp_set_num_threads(numthreads);
  double start = omp_get_wtime();

  #pragma omp parallel
  {
    int numThreads = omp_get_num_threads();
    int id = omp_get_thread_num();
    int i;
    double x;
    for(i = id; i < numSteps ; i += numThreads){
      x = (i + 0.5) * step;
      partialsum[i] = 4.0/(1.0 + x * x);
    }
  }

  for(i = 0; i < numSteps; i++)
    sum += partialsum[i];

  pi = step * sum;

  double runtime = omp_get_wtime() - start;

  printf("pi = %f in %f seconds (%i threads)\n", pi, runtime, numthreads);
}
```

Question 2 code:

```
#include "stdlib.h"
#include "stdio.h"
```

```
#include "omp.h"

static long numSteps = 100000;
double step;

int main(int argc, char* argv[]){
  argc--; argv++;
  int numthreads = 4;
  if(argc > 0) numthreads = atoi(argv[0]);

  int i;
  double pi, sum = 0.0;
  double partialsum[numSteps];

  step = 1.0/(double)numSteps;

  omp_set_num_threads(numthreads);
  double start = omp_get_wtime();

  #pragma omp parallel for reduction (+:sum)
  for(i = 0; i < numSteps ; i++){
    double x = (i + 0.5) * step;
    sum += 4.0/(1.0 + x * x);
  }

  pi = step * sum;

  double runtime = omp_get_wtime() - start;

  printf("pi = %f in %f seconds (%i threads)\n", pi, runtime, numthreads);
}
```

Question 3 code:

```
  int numthreads;
  for(numthreads = 1; numthreads <= 8; numthreads++){
    omp_set_num_threads(numthreads);

    gettimeofday(&start, NULL);
    double sum=0.0;
    #pragma omp parallel for reduction(+:sum) private(i, j, k)
    for(i=0; i<N; i++){
      for(j=0; j<N; j++){
        sum=0.0;
        for(k=0; k<N; k++){
          int a=i*N+k;
          int b=k*N+j;
          sum+=A[a]*B[b];
        }
```

```
      Cp[i*N+j]=sum;
    }
  }
  gettimeofday(&end, NULL);

  timeCost=1000000*(end.tv_sec-start.tv_sec)+(end.tv_usec-start.tv_usec);
  timeCost/=1000000;
  printf("The parallel version (%i threads) of matrix multiplication costs %lf seconds\n",
    numthreads, timeCost);
}
```

Results for first 3 questions:

question1.out
pi = 3.141593 in 0.003478 seconds (1 threads)
pi = 3.141593 in 0.002421 seconds (2 threads)
pi = 3.141593 in 0.002375 seconds (3 threads)
pi = 3.141593 in 0.002486 seconds (4 threads)
pi = 3.141593 in 0.002511 seconds (5 threads)
pi = 3.141593 in 0.002443 seconds (6 threads)
pi = 3.141593 in 0.002599 seconds (7 threads)
pi = 3.141593 in 0.002544 seconds (8 threads)

question2.out
pi = 3.141593 in 0.002860 seconds (1 threads)
pi = 3.141593 in 0.001508 seconds (2 threads)
pi = 3.141593 in 0.001073 seconds (3 threads)
pi = 3.141593 in 0.000901 seconds (4 threads)
pi = 3.141593 in 0.000931 seconds (5 threads)
pi = 3.141593 in 0.000755 seconds (6 threads)
pi = 3.141593 in 0.000816 seconds (7 threads)
pi = 3.141593 in 0.000743 seconds (8 threads)

matmul.out
Initialized successfully!
Initialized successfully!
Initialized successfully!
Initialized successfully!
The squential version of matrix multiplication costs 25.393082 seconds
The parallel version (1 threads) of matrix multiplication costs 25.788887 seconds
The parallel version (2 threads) of matrix multiplication costs 14.378166 seconds
The parallel version (3 threads) of matrix multiplication costs 9.626810 seconds
The parallel version (4 threads) of matrix multiplication costs 7.818621 seconds
The parallel version (5 threads) of matrix multiplication costs 6.255330 seconds
The parallel version (6 threads) of matrix multiplication costs 5.217202 seconds
The parallel version (7 threads) of matrix multiplication costs 4.478726 seconds
The parallel version (8 threads) of matrix multiplication costs 3.967293 seconds

question answers:

4)
No.
"j" relies on previous iterations of j.
This dependence needs to be removed.

5)
a) a is shared. The rest are private.
b) a=1, b=undef, c=3, d=undef

6)
Add #include "omp.h" to the top of the file.
Change "#pragma omp parallel" to "#pragma omp parallel private(x, i)".
Add "#pragma omp atomic" before "full_sum += partial_sum;"