امیرحسین پاشایی هیر ۹۷۳۱۰۱۳

سوال ۱.۴.c:

با pad کردن آرایه sum به اندازه cache L1 line، سرعت برنامه افزایش پیدا کرد به صورتی که زمان اجرا برای ۱ میلیارد مرحله در حالت عادی برابر با 17.569149 بوده ولی در حالت پد شده برابر با 11.074251 می‌باشد که 1.58 برابر speed up می‌باشد. کد pad شده در ضمیمه موجود است.

سوال ۲:

a) می‌توان تمام حلقه‌ها را در یک حلقه ادغام کرد، با این کار مدیریت و پخش کردن حلقه به روی نخ‌ها تنها یکبار انجام می‌شود و همچنین چون از مقدارها در زمانی که محاسبه می‌شوند استفاده می‌شوند احتمال cache hit بالاتر می‌رود.

```
#pragma omp parallel private(i)
{
    #pragma omp for reduction(+:sum)
    for(i=0;i<n;i++) {
        a[i] += b[i];
        c[i] += d[i];
        sum += a[i] + c[i];
    }
}
```

b) ابتدا مقدار دهی c را از بخش critical خارج می‌کنیم زیرا لازم نیست و c برای هر نخ خصوصی می‌باشد، همچنین می‌توان a را داخل critical section نذاشت و صرفا نوشت که به صورت جمع شدن reduction شود که مقدار نهایی درست باشد و بخش critical حذف

شود. با حذف بخش critical دیگر lock شدنی رخ نداده و سربار کم شده و سرعت برنامه افزایش می‌یابد.

```
#pragma omp parallel shared(b) private(c,d)
reduction(+:sum)
{
    ....
    a += 2*c;
    c = d*d;
}
```

c) با استفاده از single (و barrier ضمنی موجود در آن) می‌توانیم دو بخش موازی را یکی کنیم و به این صورت ساخت و مدیریت نخ‌ها توسط openmp فقط یکبار صورت می‌گیرد و برنامه سرعتش افزایش می‌یابد.

```
#pragma omp parallel
{
    #pragma omp for
    for (...) { /* Work -sharing loop 1 */ }
    #pragma omp single
    opt = opt + N; //sequential
    #pragma omp for
    for(...) { /* Work -sharing loop 2 */ }

    #pragma omp for
    for(...) { /* Work -sharing loop N */}
}
```

d) با بردن حلقه موازی به حلقه‌ی اول، ساخت thread ها و زمانبندی آن‌ها تنها یکبار انجام می‌شود و این باعث کاهش سربار اجرا می‌شود.

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

```
for (k=0; k<n; k++)
    for (j=0; j<n; j++)
        for (i=0; i<n; i++) {
        ......
        }
```

e) می‌توان حلقه را پاک کرد و جای آن یک قسمت موازی قرار داد که مقدار i همان شماره‌ی نخ می‌باشد، به این شکل کد خواناتر شده و هزینه‌ی مدیریت کردن حلقه توسط openmp نیز از برنامه حذف می‌شود.

```
int a[num_threads];
#pragma omp parallel
{
    int id = omp_get_thread_num();
    a[id] += id;
}
```

f) به منظور دوری از مدام جابجا شدن page هایی از حافظه به دلیل first touch policy، می‌توان چون آرایه‌ی هر نخ مستقل از همدیگر هستند، آرایه اصلی را تبدیل به یک آرایه یک بعدی از اشاره‌گرها کرد که این اشاره‌گرها توسط هر نخ و در نزدیکی پردازنده‌ی اجرا کننده‌ی آن نخ با آدرس آرایه‌ی هر نخ پر می‌شوند، به این شکل این جابجایی صورت نمی‌گیرد و سرعت برنامه افزایش پیدا می‌کند.

```
double *a[num_threads];
#pragma omp parallel for
for(i=0; i<num_threads; i++) {
    a[i] = (double*)malloc(N * Ń * sizeof(double));
    for(int j...)
```

```
        for(int k...)
            a[i][j * N + k] = ...
}
```

امیرحسین پاشایی هیر ۹۷۳۱۰۱۳

سوال ۳:

زمان‌های اجرای برنامه‌ی سریال:

128	512	1024
0.018327	0.527193	4.321610

جدول زمان‌ها برای موازی سازی یک بعدی به شرح زیر است:

threads\size	128	512	1024	128	512	1024
1	0.02520	0.799504	7.230956	0.72	0.66	0.60
2	0.013516	0.398704	3.397725	1.35	1.32	1.27
4	0.012529	0.342141	1.908663	1.46	1.54	2.26
8	0.006731	0.222499	1.855229	2.72	2.36	2.32

امیرحسین پاشایی هیر ۹۷۳۱۰۱۳

جدول زمان‌ها برای موازی سازی دو بعدی به شرح زیر است:

threads\size	128	512	1024	128	512	1024
1	0.02505	0.753504	6.992076	0.73	0.70	0.62
2	0.013373	0.385742	3.403440	1.37	1.36	1.27
4	0.006312	0.128940	1.290252	2.90	4.08	3.35
8	0.004698	0.121051	1.257515	3.90	4.35	2.33

تحلیل:

موازی سازی دو بعدی بهتر از یک بعدی عمل می‌کند زیرا openmp قسمت بیشتری از حلقه‌ها را در دسترس دارد برای موازی کردن و همچنین در حالت یک بعدی کل ماتریس دوم برای ضرب هر بخش خوانده می‌شد که باعث کاهش بهینگی استفاده از cache می‌شود.