پارسا محمدپور – ۹۸۲۴۳۰۵۰

امیرحسین ادواری – ۹۸۲۴۳۰۰۴

سوال اول)

- الف)

تکنیک ادغام حلقه‌ها (loop fusion) به این معنی است که چند حلقه جدا از هم را با یکدیگر یکی کنیم تا بار جابجایی حلقه‌ها (loop overhead) کاهش یابد.

برای این منظور، کد حاصل از ادغام این دو حلقه کلی، به صورت زیر خواهد بود:

```
1   for (i = 0; i < N; i++)
2   {
3       D[i] = A[i+1] * 2;
4       for (j = 0; j < N; j++)
5       {
6           if (i>0)
7               A[J] = B[i] + C[i-1];
8           else
9               A[j] = B[i];
10      }
11  }
```

توضیحات:

در این کد، در هر دو حلقه ای که در آن‌ها متغییر i از صفر تا N پیمایش می‌کند، قابل ادغام هستند؛ به طوریکه در ادغام این حلقه‌ها، حلقه اولی که به طور کلی حذف می‌شود. همچنین قبل از شروع حلقه درونی حلقه دومی، کدهای مربوط به حلقه اولی را قرار می‌دهیم.

- ب)

تکنیک بازکردن حلقه (loop unrolling) یعنی تعداد تکرارهای حلقه را با استفاده از تکرار بدنه حلقه، کاهش دهیم. این کار باعث می‌شود تا سرعت برنامه با توجه به اینکه به عناصر حلقه دسترسی سریع تری داریم و تعداد تکرار حلقه کم می‌شود؛ افزایش یابد.

برای این منظور، کد حاصل از بازکردن حلقه درونی، به صورت زیر خواهد بود:

```
for (i = 0; i < N; i++)
{
    D[i] = A[i+1] * 2;
    for (j = 0; j < N; j+=2)
    {
        if (i>0)
            A[J] = B[i] + C[i-1];
        else
            A[j] = B[i];

        if (i>0)
            A[J+1] = B[i] + C[i-1];
        else
            A[j+1] = B[i];
    }
}
```

توضیحات:

همانطور که مشخص است، بدنه حلقه درونی، دوبار تکرار شده است. یک نکته قابل توجه این اسن که در این مثال، باید حتما N، مضرب دو باشد تا این کار به این صورت قابل انجام باشد. اگر N، مضرب دو نباشد، باید شرط حلقه را به j<N-1 تغییر دهیم و در انتهای حلقه، برای j=N-1، کار مورد نیاز را انجام دهیم.(البته اگر N برابر با سایز آرایه A باشد؛ وگرنه اگر سایز آرایه A بیشتر باشد، نیازی نیست.)

اما اگر به این کد دقیق تر نگاه کنیم، می‌توانیم آن را به صورت شهودی، بهینه‌تر کنیم. صرفا با توجه به اینکه در حلقه درونی، مقدار i، تغییر نمی‌کند، می‌توانیم آن شرط ها را نیز یک بار بررسی کنیم. (البته در این حالت هم باز اینکه N مضرب دو باشد مهم می‌باشد.)

```
// More Optimized Code:
for (i = 0; i < N; i++)
{
    D[i] = A[i+1] * 2;
    for (j = 0; j < N; j+=2)
    {
        if (i>0){
            A[J+1] = B[i] + C[i-1];
            A[J] = B[i] + C[i-1];
        } else{
            A[j] = B[i];
            A[j+1] = B[i];
        }
    }
}
```

- ج)

تکنیک جداسازی حلقه (loop splitting) یعنی اینکه ما توجه به ظرط موجود در بدنه حلقه، بیایم و حلقه را به گونه‌ای تغییر دهیم که بررسی آن شرط کلا انجام نشود برای مثال در اینجا، با استفاده از این تکنیک می‌خواهیم تا شرط i>0 به ازای هر بار تکرار بدنه حلقه درونی بررسی نشود.

```
for (i = 0; i < 1; i++)
{
    D[i] = A[i+1] * 2;
    for (j = 0; j < N; j+=2)
    {
        A[j] = B[i];

        A[j+1] = B[i];
    }
}

for (i = 1; i < N; i++)
{
    D[i] = A[i+1] * 2;
    for (j = 0; j < N; j+=2)
    {
        A[J] = B[i] + C[i-1];

        A[J+1] = B[i] + C[i-1];
    }
}
```

توضیحات:

همانطور که مشخص است، حلقه های ابتدایی برای حالت else شرط می‌باشد که طبیعتا با توجه به این مورد، بدنه قسمت else نیز در آن قرار گرفته است. همچنین حلقه دومی (خط دوازده کد) برای قسمت if می‌باشد و طبیعتا بدنه همین قسمت در آن قرار گرفته است.

البته بدیهی است که می‌توان یک بهینه‌سازی دیگر در این کد انجام داد و آن این است که حلقه اولی (خط یک) را کلا حذف کنیم، زیرا تکراری نخواهد داشت. صرفا می‌توانیم به جای آن یک شرط قرار دهیم که همان شرطی است که در ابتدای حلقه چک می‌شود

```
if (0<N)
{
    D[i] = A[i+1] * 2;
    for (j = 0; j < N; j+=2)
    {
        A[j] = B[0];

        A[j+1] = B[0];
    }
}

for (i = 1; i < N; i++)
{
    D[i] = A[i+1] * 2;
    for (j = 0; j < N; j+=2)
    {
        A[J] = B[i] + C[i-1];

        A[J+1] = B[i] + C[i-1];
    }
}
```

- د)

تکنیک بلوک کردن حلقه (loop blocking) به این صورت است که به جای اینکه حلقه را به صورت کامل پیمایش کنیم، آن را به بلوک‌هایی تقسیم می‌کنیم و سپس هر بلوک را پیمایش می‌کنیم و بعد از اتمام هر بلوک به سراغ بلوک بعدی می‌رویم. این کار باعث افزایش سرعت می‌شود.

```
int block_size = 16;

for (i = 0; i < 1; i++)
{
    D[i] = A[i+1] * 2;
    for (jj = 0; jj < N; jj+=block_size)
    {
        for (j = jj; j < min(jj+block_size-1, N); j++)
        {
            A[j] = B[i];

            A[j+1] = B[i];
        }
    }
}

for (i = 1; i < N; i++)
{
    D[i] = A[i+1] * 2;
    for (jj = 0; jj < N; jj+=block_size)
    {
        for (j = jj; j < min(jj+block_size-1, N); j++)
        {
            A[J] = B[i] + C[i-1];

            A[J+1] = B[i] + C[i-1];
        }
    }
}
```

توضیحات:

همانطور که از ما خواسته شده است، برای حلقه درونی، این تکنیک را برای بلوک‌هایی با اندازه ۱۶ انجام می‌دهیم. برای این کار مطابق با اسلایدهای درس، ابتدا یک حلقه قرار می‌دهیم که کل بلوک ها را پیمایش می‌کند (حلقه‌ای که متغییر آن jj می‌باشد) سپس یک حلقه درونی قرار می‌دهیم (که متغیر درونی آن j است) که این حلقه درونی، بر روی هر بلوک پیمایش را انجام می‌دهد. (البته همانطور که در قسمت قبل گفته شده بود، می‌توانیم کلا حلقه موجود در خط یک را برداریم و به جای آن یا یک شرط بگذاریم.)