پارسا نوری
عرفان رفیعی
سری هفتم امبدد - گروه ۹

الف)

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

من این فیوژن را با حرکت حلقه اول داخل حلقه دوم و قرار دادن آن قبل از حلقه داخلی انجام دادم. به این ترتیب، متغیر حلقه i توسط هر دو حلقه مشترک می شود و تعداد تکرارها از N2 به N کاهش می یابد.
این ادغام امکان پذیر است زیرا حلقه اول به حلقه دوم بستگی ندارد یا بالعکس. حلقه اول فقط از A می خواند و به D می نویسد، در حالی که حلقه دوم فقط از B و C می خواند و به A می نویسد. بنابراین هیچ تضاد یا تداخلی بین حلقه ها وجود ندارد. ادغام همچنین عملکرد حافظه نهان را بهبود می بخشد زیرا عناصر داده در زمان و مکان با دقت بیشتری در دسترس قرار می گیرند، که باعث کاهش کمبود حافظه نهان و پهنای باند حافظه می شود.

ب)

Loop unrolling تکنیکی است که سرعت اجرای برنامه را با کاهش یا حذف دستورالعمل‌هایی مانند محاسبات pointer و تستهای پایان حلقه در هر تکرار که حلقه را کنترل میکنند، بهینه میکند. تبدیل می تواند به صورت دستی توسط برنامه نویس یا توسط یک کامپایلر بهینه سازی انجام شود. باز کردن حلقه همچنین می تواند کارایی برنامه را با اجازه دادن به اجرای parallel دستورات حلقه افزایش دهد.
کد مورد نظر :

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

پارسا نوری
عرفان رفیعی
سری هفتم امبدد - گروه ۹

ج)

در این قسمت با استفاده از Loop splitting اقدام به بهینه‌سازی کد می‌نماییم به این صورت که if که در کد وجود دارد باعث می‌شود که بهینگی کد کاهش یابد زیرا این کد به ازای هر i , j یعنی N^2 بار باید تکرار شود پس حذف آن می‌تواند باعث می‌شود که کد بهینه‌تر باشد. حال می‌توان for درونی را به ازای i=0 به بیرون می‌آوریم. سپس می‌توانیم if را از حلقه درونی حذف نماییم.

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

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

د)

یک تکنیک بهینه سازی Loop tiling/blocking locality برنامه است که هدف آن بهبود است. الگوی دسترسی به حافظه reference حلقه را به منظور استفاده مجدد از داده های قبل از خروج داده ها و cache موجود در حافظه نیاز به بارگیری مجدد از حافظه، تغییر می دهد در این کد، حلقه داخلی به بلوک هایی با اندازه 16 (یا کوچکتر اگر عناصر باقی مانده کمتر از 16 باشند) تقسیم می شود. لازم به ذکر است که حلقه موجود در حلقه داخلی نهایتا تا کمینه j + block_size و N می‌رود.

```
for (j=0; j < N; j++){
    A[j] = B[0];
    A[j] = B[1] + C[0];
}

int block_size = 16;
for (i=2; i < N; i++){
    D[i] = A[i+1] * 2;
    for (j=0; j < N; j += block_size){
        for (int k=j; k < min(j + block_size, N) ; k++){
            A[j] = B[i] + C[i-1];
            A[j] = B[i+1] + C[i];
        }
    }
}
```