**Ödevi yapanlar:**

Fatihcan Uslu

**Problem tanımı:**

Ardışık bir şekilde yani her pivot’un sol ve sağ düğümünün sırayla çalışmasını bekleyen quicksort algoritmasının daha hızlı çalışabilmesi için bu düğümlerin aynı anda birbirlerini etkilemeden ve race condition durumuna düşmeden paralel olarak çalıştırılması.

**Kullanılan yöntem:**

Seçilen yöntemde belirli sayıda process kullanılarak quicksort algoritmasındaki pivot’ların sol ve sağ düğümleri çalıştırılarak farklı bilgisayarların aynı anda çalıştırılması sağlanmıştır, bu sayede yapılacak iş process’lere bölünmüş ve paralelleştirilmiştir. Process sayısı sınırlandırılmıştır çünkü büyüyen indis boyutlarında sınırlandırılmamış process sayısına sahip olmak hızlandırmaktan ziyade programın yavaşlamasına sebep olmaktadır. Çünkü process oluşturma işlemide bir maliyettir ve process’lerin yapması gereken işten daha üzün sürebilmektedir. Bu yüzden tanımlanan “procNum” değişkeni aracılığı ile process sayısı kontrol edilmektedir. Paralelleştirme için thread yerine process kullanıldı. Çünkü procNum değişkeni için thread kullanılsaydı değerin kontrol edilmesi gerekilecekti. Bunun sebibi her bir process, kendi belleğinde izole edilmiş değişkenlere sahipken, aynı process içindeki threadler bu değişkenleri paylaşarak erişebilir. Yani thread’ler içinde oluşturulan diğer threadlerin sayısını kontrol etmek sorun yaratabilmekteydi. Bu sorun aslında değişkenin global hale getirilmesi yada threadlerin birbirlerine en azından bu değişken için senkronize olacak şekilde bağlanmasıyla çözülebilir.

Belirli sayıda process oluşturulduktan sonra kod ardışık quicksort gibi çalıştırılmaktadır.

**Paralel çalışan kod içinde ardışık çalışan kodun paralel çalışan koda göre hız bakımından oranı nedir:**

Tamamıyla paralelleştirilen kod kısmen paralelleştirilen kod’a göre kıyaslandığında daha yavaş çalıştığı hatta işlemin sonuçlanmadığı gözlenmiştir. Kodun tamamen paralelleştirilmesi istenilen bir durum olsada paralel kısımların çözüm üreteceği kısmın hesaplama maliyeti yüksek olmalı yoksa paralelleştirme hız artışı sağlayamamaktadır.

**Ardışık çalışan ve parallel çalışan versiyonu girdi boyutu, paralel çalışan thread/bilgisayar sayısına göre hız ve karmaşık bakımından kıyaslanması:**

Girdi boyutu çok küçük seçildiğinde (100000’den daha az) genel olarak ardışık algoritmanın daha hızlı olduğu gözlenmiştir.

* Ardışık çalışma zamanı: 0 saniye
* Paralel çalışma zamanı: 0.5 saniye

Girdi boyutu çok yüksek seçildiğinde (1000000’den daha fazla) genel olarak paralel algoritmanın ardışık algoritmaya göre 2 kat daha hızlı olduğu gözlenmiştir.

* 2 milyon için ardışık çalışma zamanı: 16.11 saniye
* 2 milyon için Paralel çalışma zamanı: 9.82 saniye

Girdi boyutu sabit tutulup process sayısı arttırıldığında paralel çalışma hızında büyük farklılıklar gözlenmemiştir. Aşağıdaki örnek 15 process’ten 31 process’e geçişteki paralel hız değişimini ifade etmektedir.

* 1 process için paralel çalışma zamanı: 6.16 saniye
* 3 process için paralel çalışma zamanı: 5.62 saniye
* 15 process için paralel çalışma zamanı: 3.26 saniye
* 63 process için paralel çalışma zamanı: 3.46 saniye
* Limitsiz process için paralel çalışma zamanı: Çalışmadı

**Önemli Not:**

Paralel çalışan kısım için pivot seçimi olabilecek en basit yöntem olan indisin ilk elemanı mantığına dayanmaktadır. Bu process’lerin boyutlandırılmasında farklılıklara sebep olmaktadır. Yani bazı process’lere atanan dizilerin boyutları diğerlerinden daha kısa yada uzun olabilmektedir. Buda bazı processlerin daha erken bitmesi anlamına gelmektedir. Ancak bir sistem en zayıf halkası kadar hızlıdır ve en geç çalışan process sonlanmadan işlem tamamlanamaz. Bunun önüne geçebilmek için process’lere birbirlerine yakın boyutlarda diziler atanabilmesi için bir çalışma yapılırsa hız artışı gözlenebilir.

Daha fazla açıklama ve test sonucu için kodun çalışma mantığını anlatan youtube videosu incelenebilir:  
Youtube videosu linki: <https://youtu.be/SgfDtH5dT9I>

Github linki: <https://github.com/FatihcanUslu/Parallel_Quicksort>