

BÖL VE YÖNET ALGORİTMALARININ ENERJİ TÜKETİMİ VE KARBON EMİSYONU AÇISINDAN KARŞILAŞTIRMALI ANALİZİ

A COMPARATIVE ANALYSIS OF DIVIDE-AND-CONQUER ALGORITHMS IN TERMS OF ENERGY CONSUMPTION AND CARBON EMISSIONS

Damla AKPINAR^{*1} Ülkü Bensu İNCE¹ Onur İLGİN¹ Necip GÖZÜAÇIK²

¹ Bilgisayar Mühendisliği Bölümü, Mühendislik ve Doğa Bilimleri Fakültesi, İstanbul Sağlık ve Teknoloji Üniversitesi, İstanbul, Türkiye.

² Yazılım Mühendisliği Bölümü, Mühendislik ve Doğa Bilimleri Fakültesi, İstanbul Sağlık ve Teknoloji Üniversitesi, İstanbul, Türkiye.

ÖZET

Bu çalışmada, böl ve yönet yaklaşımına dayalı MergeSort ve QuickSort sıralama algoritmaları ile klasik matris çarpımı ve Strassen algoritmaları çalışma süresi, tahmini işlemci enerji tüketimi ve karbon emisyonu açısından deneysel olarak karşılaştırılmıştır. Algoritmalar saf Python (3.11.9) ile gerçekleştirilmiş, işlemci enerjisi CodeCarbon kütüphanesinin işlemci yüküne dayalı modeliyle, yaklaşık 10 saniyelik toplu çalıştırma pencereleri kullanılarak tahmin edilmiştir. Sıralama deneylerinde beş veri tipi (rastgele, sıralı, ters sıralı, kısmen sıralı ve yüksek tekrar oranlı) ve 10.000–500.000 eleman aralığında beş boyut, matris deneylerinde 64×64–512×512 boyutları kullanılmış; her koşul 20 tekrarla çalıştırılmış ve eşleştirilmiş Wilcoxon testi, etki büyüklüğü ve Holm düzeltmesiyle değerlendirilmiştir. Rastgele veride iki sıralama algoritması arasında tutarlı ve anlamlı bir enerji farkı gözlenmemiştir. Diğer dört veri tipinde üç yollu bölümlleme kullanan QuickSort uygulaması MergeSort’a göre anlamlı düzeyde daha az enerji tüketmiş; fark yüksek tekrar oranlı veride %67–76 düzeyine ulaşmıştır. Eşik değeri 16 olan hibrit Strassen uygulaması, test edilen tüm boyutlarda üç döngülü klasik çarpıma göre daha kısa çalışma süresi ve daha düşük enerji tüketimi sergilemiş; enerji farkı 512×512 boyutunda %47 olmuştur. Bulgular, enerji davranışının asimptotik karmaşıklığın yanında veri dağılımına ve uygulama tercihlerine de bağlı olduğunu göstermekte olup test edilen koşullar ve değerlendirilen uygulamalarla sınırlıdır.

Anahtar Kelimeler: Böl ve Yönet Algoritmaları, CodeCarbon, Enerji Karmaşıklığı, Karbon Emisyonu, Performans Analizi, Yeşil Bilişim.

ABSTRACT

This study presents a comparative analysis of four fundamental divide-and-conquer algorithms—MergeSort, QuickSort, conventional matrix multiplication, and Strassen matrix multiplication—in terms of execution time, energy consumption, and carbon emissions. The experiments were conducted in a Python environment using the CodeCarbon library. Random datasets and datasets containing a high proportion of duplicate elements, ranging from 10,000 to 500,000 elements, were used for the sorting algorithms, whereas square matrices ranging in size from 64 × 64 to 256 × 256 were used for the matrix multiplication algorithms. The results demonstrated that QuickSort exhibited lower energy consumption and carbon emissions than MergeSort, particularly for large datasets containing a high proportion of duplicate elements. In the matrix multiplication experiments, despite its theoretical advantage, the Strassen algorithm consumed more energy than the conventional method because of the memory overhead associated with its recursive structure. These findings indicate that algorithm selection should consider not only theoretical time complexity but also energy efficiency and environmental impact, emphasizing the importance of sustainable software development.

Keywords: Carbon Emissions, CodeCarbon, Divide-and-Conquer Algorithms, Energy Complexity, Green Computing, Performance Analysis.

*Corresponding Author (Sorumlu Yazar), e-mail: damlaakpinaaar@gmail.com

1. GİRİŞ

Son yıllarda yazılım sistemlerinin enerji tüketimi ve çevresel etkileri, performans değerlendirme çalışmalarında giderek daha önemli bir araştırma konusu hâline gelmiştir. Özellikle yüksek işlem gücü gerektiren uygulamaların yaygınlaşmasıyla birlikte yalnızca çalışma süresi değil, enerji tüketimi ve karbon emisyonu gibi ölçütler de yazılım performansının değerlendirilmesinde dikkate alınmaya başlanmıştır [1,2]. Bu durum, enerji verimliliğini yalnızca donanım düzeyinde değil algoritma tasarımı ve çalışma zamanı optimizasyonları açısından da incelemeyi gerekli kılmaktadır.

Yeşil Bilişim ve enerji farkındalıklı yazılım geliştirme çalışmaları, algoritma seçimi, kaynak kullanımı ve yürütme stratejilerinin enerji tüketimi üzerinde belirgin etkiler oluşturabileceğini göstermektedir [3,4]. Özellikle aynı işlevi yerine getiren algoritmaların farklı işlem sayısı, bellek erişim düzeni ve çalışma zamanı davranışı nedeniyle farklı enerji profilleri oluşturabildiği bildirilmektedir [5]. Bu nedenle algoritmaların yalnızca teorik zaman karmaşıklığı ile değil, gerçek çalışma koşullarında ortaya çıkan enerji tüketimiyle birlikte değerlendirilmesi gerekmektedir.

Sıralama algoritmaları ve matris çarpımı işlemleri hem yaygın kullanımları hem de hesaplama maliyetleri nedeniyle enerji verimliliği analizlerinde sık tercih edilen problem sınıflarıdır. QuickSort ve MergeSort gibi sıralama algoritmaları farklı veri düzenleri altında değişen performans ve enerji davranışları gösterebilmektedir [5]. Benzer şekilde klasik matris çarpımı ile Strassen algoritması arasındaki farklar, yalnızca işlem süresi açısından değil, bellek kullanımı ve enerji tüketimi açısından da önem taşımaktadır [6,7].

Algoritmaların enerji davranışının güvenilir biçimde analiz edilebilmesi için yazılım tabanlı ölçüm yöntemlerinin doğru kullanılması gerekir. İşlemci düzeyinde enerji ölçümü yapan yazılım tabanlı araçlar, algoritmaların gerçek çalışma ortamındaki güç ve enerji davranışlarını incelemek için yaygın olarak kullanılmaktadır [8]. Bu tür ölçümler, teorik karmaşıklık analizinin deneysel enerji profiliyle birlikte yorumlanmasına olanak sağlamaktadır.

Python çalışma zamanı ortamı da bu bağlamda önemli bir değişken olarak değerlendirilebilir. Python kodlarının farklı çalışma ortamlarında, yürütme biçimlerinde ve yorumlayıcı yapılandırmalarında farklı süre, bellek ve güç tüketimi değerleri üretebildiği gösterilmiştir [9,10]. Bu nedenle Python tabanlı algoritma deneylerinde yalnızca algoritmanın kendisi değil, kullanılan çalışma zamanı sürümü ve yürütme ortamı da dikkate alınmalıdır.

Bu çalışmada QuickSort, MergeSort, klasik matris çarpımı ve Strassen algoritmalarının farklı veri boyutları ve girdi dağılımları altındaki çalışma süreleri, tahmini işlemci enerji tüketimleri ve karbon emisyonları karşılaştırmalı olarak incelenmiştir. Deneyler tek bir çalışma zamanı ortamında (Python 3.11.9) yürütülmüş; yazılım tabanlı bir enerji tahmin aracı, toplu ölçüm (batch) protokolü ve eşleştirilmiş parametrik olmayan istatistiksel testler kullanılmıştır.

2. MATERYAL VE METOT

2.1. Kullanılan Algoritmalar ve Uygulama Ayrıntıları

Bu çalışmada, enerji tüketimi ve hesaplama performansı davranışlarının karşılaştırılması amacıyla iki farklı sıralama algoritması ile iki farklı matris çarpımı algoritması incelenmiştir. Algoritmaların gerçek çalışma ortamındaki davranışlarının değerlendirilebilmesi için uygulama ayrıntıları, veri yapıları ve kullanılan taban durum koşulları dikkate alınmıştır. Kullanılan algoritmaların uygulama özellikleri aşağıda açıklanmıştır.

2.1.1. MergeSort

MergeSort algoritması, iki yönlü böl ve yönet yaklaşımı kullanılarak uygulanmıştır. Dizi uzunluğu bir veya daha az olduğunda özyineleme sonlandırılmaktadır. Daha büyük diziler orta noktadan iki alt diziye ayrılmakta, alt diziler özyinelemeli olarak sıralanmakta ve daha sonra birleştirme yordamı kullanılarak tek bir sıralı dizi elde edilmektedir.

Birleştirme aşamasında iki sıralı alt dizi karşılaştırılarak küçük olan eleman sonuç dizisine aktarılmakta, alt dizilerden birinin elemanları tükendiğinde diğer alt dizide kalan elemanlar doğrudan sonuç dizisine eklenmektedir. Eşit elemanlar karşılaştırılırken sol alt dizideki elemanın önce seçilmesi nedeniyle

uygulama kararlı sıralama davranışı göstermektedir. Uygulamada alt dizilerin oluşturulması ve birleştirme sonucunun tutulması için yeni Python listeleri kullanılmaktadır.

2.1.2. QuickSort

QuickSort algoritması, tekrarlı elemanların bulunduğu veri kümelerinde oluşabilecek gereksiz özyinelemeyi azaltmak amacıyla üç liste tabanlı bir ayrıştırma yaklaşımıyla uygulanmıştır. Pivot olarak dizinin orta indeksindeki eleman seçilmektedir. Daha sonra giriş dizisi; pivottan küçük elemanları içeren sol, pivota eşit elemanları içeren orta ve pivottan büyük elemanları içeren sağ listelerine ayrılmaktadır. sol ve sağ listeleri özyinelemeli olarak sıralanmakta ve sonuç `quick_sort(sol) + orta + quick_sort(sağ)` biçiminde birleştirilmektedir. Pivot değerine eşit olan elemanların middle listesinde tutulması, özellikle tekrar oranı yüksek veri kümelerinde bu elemanların yeniden özyinelemeli olarak işlenmesini önlemektedir. Dizi uzunluğu bir veya daha az olduğunda algoritma taban duruma ulaşarak özyinelemeyi sonlandırmaktadır.

Bu uygulama, yerinde (in-place) bir üç yollu bölümleme gerçekleştirmemekte; üç ayrı Python listesi oluşturularak ayrıştırma yapmaktadır. Bu nedenle her bölümleme adımında sol, orta ve sağ listeleri için ek bellek ayrılmaktadır; bellek kullanımı bu çalışmada ölçülmemiştir.

Uygulamada küçük alt diziler için ayrı bir sıralama eşiği (cutoff) veya özel bir özyineleme derinliği sınırı kullanılmamış, özyineleme yalnızca taban durum koşulunda ($n \leq 1$) sonlandırılmıştır.

2.1.3. Klasik Matris Çarpımı

Klasik matris çarpımı, kütüphane tabanlı optimize matris çarpımı işlemlerinden bağımsız olarak saf Python üçlü döngüsü kullanılarak gerçekleştirilmiştir. Sonuç matrisi, giriş matrisinin veri tipiyle aynı veri tipinde sıfırlardan oluşan bir NumPy dizisi olarak oluşturulmuştur. Deneylerde kullanılan giriş matrisleri int64 veri tipindedir.

Çarpım işlemi, standart matris çarpımı formülasyonuna uygun olarak üç iç içe döngü ile gerçekleştirilmiştir. Dış döngü satırları, ikinci döngü sütunları ve iç döngü ise ilgili satır-sütun elemanlarının çarpımlarının toplanmasını kontrol etmektedir.

2.1.4. Strassen Matris Çarpımı ve Taban Durum Eşiği

Strassen algoritması, iki matrisin çarpımındaki klasik sekiz alt matris çarpımını yedi özyinelemeli çarpıma indirerek işlem sayısını azaltan böl ve yönet yaklaşımıyla uygulanmıştır [11]. Algoritmanın teorik zaman karmaşıklığı $O(n^{\log_2 7}) \approx O(n^{2.81})$ düzeyindedir.

Uygulamada, küçük boyutlu alt matrislerde özyinelemeli çağrılar ve ara işlemlerin oluşturabileceği ek maliyeti sınırlandırmak amacıyla 16 değerinde bir taban durum eşiği (cutoff) kullanılmıştır. Matris boyutu 16 veya daha küçük olduğunda özyineleme sonlandırılmakta ve klasik matris çarpımı yöntemine geçilmektedir. Eşik değeri, 128×128 , 256×256 ve 512×512 boyutlarında 16, 32, 64 ve 128 eşiklerinin üçer tekrarla karşılaştırıldığı bir ön deneme sonucunda belirlenmiştir; üç boyutta da en kısa süre eşik 16 ile elde edilmiştir (512×512 için ortalama 28,8 s; eşik 128 ile 41,6 s).

Strassen algoritmasında matrisler dört alt bloğa NumPy dilimleme (slicing) işlemiyle ayrılmaktadır. Alt blokların oluşturulması başlangıçta NumPy görünümü (view) niteliğinde olmakla birlikte, alt blokların toplanması ve çıkarılması sırasında yeni geçici NumPy dizileri oluşturulmaktadır. Yedi özyinelemeli çarpımın sonucunda dört sonuç bloğu hesaplanmakta ve bu bloklar `np.hstack` ve `np.vstack` fonksiyonları kullanılarak nihai sonuç matrisi hâline getirilmektedir.

Deneylerde Strassen algoritmasına verilen matrisler int64 veri tipinde oluşturulmuştur ve ara matris işlemleri bu veri tipi üzerinden gerçekleştirilmiştir. Algoritmanın doğruluğu, enerji ve çalışma süresi ölçüm penceresinin dışında, elde edilen sonuç ile referans NumPy matris çarpımının `np.array_equal` fonksiyonu kullanılarak karşılaştırılması yoluyla kontrol edilmiştir.

2.2. Deney Ortamı

Çalışma kapsamında gerçekleştirilen tüm deneyler aynı donanım ve yazılım ortamında yürütülmüştür. Deneylerin aynı sistem üzerinde gerçekleştirilmesiyle algoritmalar arasındaki performans ve enerji

farklılıklarının donanım değişkenlerinden etkilenmesinin azaltılması amaçlanmıştır. Deneylerde tek bir Python çalışma ortamı kullanılmış ve farklı Python sürümleri arasında karşılaştırma yapılmamıştır.

Deneysel çalışmalarda Python 3.11.9 (64-bit, AMD64) kullanılmıştır. Algoritmaların uygulanmasında Python'un standart veri yapıları ve kontrol yapıları kullanılmış; matris işlemleri ve veri işleme süreçlerinde NumPy, sonuçların düzenlenmesi ve veri analizi aşamalarında Pandas, grafiklerin oluşturulmasında ise Matplotlib kullanılmıştır. İşlemci enerji tüketimi CodeCarbon kütüphanesinin 3.2.6 sürümü ile tahmin edilmiştir. Kullanılan yazılım bileşenlerinin sürümleri ile deneylerin gerçekleştirildiği donanım ve sistem koşulları Tablo 1'de özetlenmiştir. Sistem AC güç bağlantısında çalıştırılmış, Windows güç modu Balanced (dengeli) olarak ayarlanmış ve Energy Saver kapalı tutulmuştur. İşlemcinin turbo çalışma politikası sistem/ürün yazılımı (firmware) varsayılana bırakılmış, işlemci sıcaklığı izlenmemiştir.

Tablo 1. Deneylerde kullanılan donanım ve yazılım ortamı

Bileşen	Deney ortamı
İşletim sistemi	Windows 11 Home Single Language 25H2 (Build 26200.9457)
İşlemci	13th Gen Intel Core i7-13700H
Fiziksel çekirdek	14
Mantıksal işlemci	20
Bellek	16 GB (2 × 8 GB Samsung DDR4-3200)
Güç durumu	AC güç bağlantısı
Windows güç modu	Balanced
Energy Saver	Kapalı
CPU turbo politikası	Sistem/ürün yazılımı varsayılana
Python	3.11.9, 64-bit AMD64
NumPy	2.4.4
Pandas	3.0.3
Matplotlib	3.10.9
CodeCarbon	3.2.6
psutil	7.2.2

Deneylerin tekrarlanabilirliğini sağlamak amacıyla veri üretiminde sabit başlangıç tohumları kullanılmıştır. Deney akışında her veri tipi ve veri boyutu için tanımlı tohum yapısı kullanılarak veri kümelerinin yeniden oluşturulabilmesi sağlanmıştır.

2.3. Veri Kümeleri ve Giriş Boyutları

Algoritmaların farklı giriş koşullarındaki çalışma süresi ve enerji davranışlarını inceleyebilmek amacıyla sıralama ve matris çarpımı deneyleri için farklı veri kümeleri oluşturulmuştur. Veri üretimi deney kodu içerisinde otomatik olarak gerçekleştirilmiş ve tekrar üretilebilirlik amacıyla Python random modülü ile NumPy rastgele sayı üreticisi için sabit tohumlar kullanılmıştır.

Sıralama algoritmaları için rastgele, sıralı, ters sıralı, kısmen sıralı ve yüksek tekrar oranlı olmak üzere beş farklı veri tipi kullanılmıştır. Veri boyutları 10.000, 50.000, 100.000, 250.000 ve 500.000 eleman olarak belirlenmiştir.

Rastgele veri kümelerinde elemanlar 0–1.000.000 aralığında rastgele tamsayılar olarak üretilmiştir. Sıralı veri kümeleri 0'dan başlayarak artan biçimde oluşturulmuş, ters sıralı veri kümelerinde ise değerler n'den 1'e doğru azalan biçimde düzenlenmiştir. Kısmen sıralı veri kümeleri başlangıçta artan sıralı olarak oluşturulmuş ve eleman sayısının %5'inin tam sayı kısmı kadar rastgele yer değiştirme işlemi uygulanmıştır. Yüksek tekrar oranlı veri kümelerinde ise değerler 0–100 aralığındaki rastgele tamsayılardan oluşturulmuştur.

Her veri tipi ve veri boyutu için kullanılan tohum, deney koşullarına göre belirlenmiş ve her tekrar için farklılaştırılmıştır. Sıralı ve ters sıralı veri kümeleri tohumdan bağımsız, deterministik biçimde üretildiğinden bu iki veri tipinde tüm tekrarlar aynı girdi üzerinde yapılmıştır. Böylece aynı deney

koşulunda veri üretiminin kontrol edilebilir ve yeniden üretilebilir olması sağlanmıştır. Her ölçümde aynı veri kümesinin kopyaları algoritmalarla ayrı ayrı verilerek algoritmaların birbirlerinin girdilerini değiştirmesi engellenmiştir.

Matris çarpımı deneylerinde 64×64 , 128×128 , 256×256 ve 512×512 boyutlarında kare matrisler kullanılmıştır. Matris elemanları 0–99 aralığında rastgele tamsayılar olarak üretilmiş ve NumPy int64 veri tipinde tutulmuştur. Her matris boyutunda iki algoritma aynı giriş matrisleri üzerinde çalıştırılmıştır. Algoritmaların doğruluğu, elde edilen sonuçların NumPy’nin doğrudan matris çarpımı işlemi ile oluşturulan referans sonuçla np.array_equal kullanılarak karşılaştırılmasıyla kontrol edilmiştir.

Tüm deney koşulları 20 tekrar olacak şekilde yürütülmüştür. Tekrarlar arasında aynı deney koşulunun korunması ve veri üretiminin yeniden oluşturulabilmesi için deney tohumları kayıt altına alınmıştır.

2.4. Çalışma Süresi Ölçümü ve Enerji Tahmini

Algoritmaların çalışma süresi ölçümlerinde Python’un time.perf_counter() fonksiyonu kullanılmıştır. Çok kısa süreli tek bir çalıştırmanın enerji tahmininde oluşturabileceği belirsizliği azaltmak amacıyla enerji tahminleri tek bir çalıştırma yerine toplu çalıştırma (batch) yaklaşımıyla elde edilmiştir.

Her deney koşulunda ölçümden önce algoritma iki kez ısıtma çalıştırmasına tabi tutulmuştur. Ardından kısa bir pilot çalışma gerçekleştirilerek algoritmanın yaklaşık çalışma süresi belirlenmiş ve toplu ölçümde kullanılacak tekrar sayısı hesaplanmıştır. Toplu ölçüm penceresinin yaklaşık 10 saniye sürmesi hedeflenmiş, ancak bir pencere içerisindeki tekrar sayısı en fazla 10.000 ile sınırlandırılmıştır. Tekrar sayısı pilot süreye göre belirlendiğinden gerçekleşen pencere süreleri koşullar arasında değişmiş (sıralama deneylerinde medyan yaklaşık 10 s), tek çalıştırması 10 saniyeyi aşan 512×512 matris çarpımında ise pencere tek çalıştırmadan oluşmuştur. Ölçülen toplam batch süresi tekrar sayısına bölünerek tek çalıştırmaya karşılık gelen ortalama çalışma süresi elde edilmiştir.

İşlemci enerji tüketiminin tahmininde CodeCarbon 3.2.6 kullanılmıştır. Deney bilgisayarında Intel RAPL sayaçlarına doğrudan erişim bulunmadığından CodeCarbon işlemci enerjisini donanım sayacından ölçmemiş, işlemci yüküne dayalı modelle (cpu_load) tahmin etmiştir. İşlemci modeli aracın güç veri tabanında yer almadığından bu modelde iş parçacığı başına 4 W varsayılan değeri (20 iş parçacığı için 80 W) kullanılmıştır. Bu nedenle çalışmada raporlanan enerji değerleri doğrudan ölçüm değil, tahmini değerlerdir; bu yaklaşımın sonuçlara etkisi Bölüm 3.4’te tartışılmıştır. CodeCarbon takipçisi yalnızca toplu çalışma süresini kapsayacak şekilde başlatılmış ve batch tamamlandıktan sonra durdurulmuştur. Temel enerji ölçütü olarak CodeCarbon’un cpu_energy değeri kullanılmış; batch boyunca elde edilen CPU enerji değeri batch içerisindeki tekrar sayısına bölünerek tek çalıştırmaya karşılık gelen tahmini CPU enerji tüketimi hesaplanmıştır.

Buna ek olarak CodeCarbon tarafından sağlanan energy_consumed değeri de toplam takip edilen enerji olarak kaydedilmiştir. Bu değer de batch tekrar sayısına bölünerek tek çalıştırma başına karşılık gelen değer elde edilmiştir. Böylece CPU enerji tüketimi ile CodeCarbon tarafından takip edilen toplam enerji ayrı değişkenler olarak korunmuştur.

Karbon emisyonu doğrudan çalışma süresinden veya ortalama işlemci gücünden hesaplanmamıştır. Karbon emisyonu, CodeCarbon’un konuma dayalı emisyon tahmini yerine, tek çalıştırmaya ait tahmini CPU enerji tüketiminin Türkiye için yayımlanan 2023 yılı dağıtım şebekesine bağlı tüketim noktası emisyon faktörüyle ($0,469 \text{ kgCO}_2\text{e/kWh}$) çarpılmasıyla hesaplanmıştır [12]:

$$\text{CO}_2\text{e} = \text{CPU Energy} \times 0,469 \quad (1)$$

Burada CO_2e , tek çalıştırmaya karşılık gelen karbon emisyonunu kgCO_2e cinsinden; CPU Energy, tek çalıştırmaya karşılık gelen CPU enerji tüketimini kWh cinsinden; 0,469 ise kullanılan karbon dönüşüm katsayısını $\text{kgCO}_2\text{e/kWh}$ cinsinden ifade etmektedir.

Ölçümlerde algoritmaların çalıştırılma sırası da dengelenmiştir. Tek numaralı tekrarlar algoritmaların bir sırasıyla, çift numaralı tekrarlar ise ters sırayla gerçekleştirilmiştir. Böylece sabit bir algoritma sırasından kaynaklanabilecek sistematik etkilerin azaltılması amaçlanmıştır. Aynı deney koşulundaki algoritmalar için eşdeğer giriş verileri kullanılmış ve her algoritmaya ölçüm sırasında ayrı veri kopyası verilmiştir.

Algoritmaların doğruluk kontrolleri her deney koşulunun ilk tekrarında, ölçüm penceresinin dışında gerçekleştirilmiştir; sıralama sonuçları Python'un sorted() fonksiyonunun çıktısıyla, matris çarpımı sonuçları ise NumPy'nin A @ B sonucuyla karşılaştırılmıştır. Her deney koşulundan elde edilen ham ölçüm kayıtları CSV formatında saklanmış ve daha sonraki istatistiksel analizlerde kullanılmıştır. Sıralama ve matris çarpımı deneylerinin her biri 20 tekrar üzerinden değerlendirilmiştir.

2.5. İstatistiksel Analiz

Deney sonuçlarının değerlendirilmesinde çalışma süresi ve CPU enerji tüketimi temel ölçütler olarak kullanılmıştır. Her deney grubu için gözlem sayısı (n), aritmetik ortalama, örneklem standart sapması (SS), medyan, minimum ve maksimum değerler ile değişim katsayısı (CV) hesaplanmıştır. Ayrıca ortalamalar için t-dağılımına dayalı %95 güven aralıkları hesaplanmıştır.

Algoritmalar arasındaki karşılaştırmalarda deneylerin eşleştirilmiş yapısı dikkate alınmıştır. Eşleştirme; veri tipi, veri boyutu, tekrar numarası ve deney tohumu bilgileri üzerinden yapılmıştır. Her karşılaştırmada 20 eşleştirilmiş gözlem bulunması şartı uygulanmıştır. Bu yapı, aynı deney koşulunda elde edilen algoritma sonuçlarının doğrudan karşılaştırılmasına olanak sağlamıştır.

İki algoritmanın çalışma süresi ve CPU enerji tüketimi açısından karşılaştırılmasında iki yönlü Wilcoxon işaretli sıralar testi kullanılmıştır. İstatistiksel anlamlılık düzeyi 0,05 olarak belirlenmiştir. Etki büyüklüğünün değerlendirilmesinde eşleştirilmiş rank-biserial korelasyon katsayısı (r) kullanılmıştır; negatif r değerleri QuickSort veya Strassen algoritmasının daha düşük değer ürettiğini göstermektedir.

Birden fazla deney koşulunda gerçekleştirilen istatistiksel karşılaştırmalarda çoklu karşılaştırma kaynaklı hata olasılığını kontrol etmek amacıyla Holm düzeltmesi uygulanmıştır. Düzeltme, çalışma süresi ve CPU enerji tüketimi karşılaştırmaları için ayrı test aileleri üzerinden (sıralama deneylerinde her ölçüt için 25, matris deneylerinde 4 karşılaştırma) gerçekleştirilmiştir. Düzeltilmiş p-değeri 0,05'in altında olduğunda sonuç istatistiksel olarak anlamlı kabul edilmiştir.

Karbon emisyonu, CPU enerji tüketiminden türetilen bir ölçüt olduğundan temel istatistiksel karşılaştırmalarda bağımsız birincil test değişkeni olarak kullanılmamıştır. İstatistiksel analizler ham tekrar sonuçları üzerinden gerçekleştirilmiş, yalnızca ortalama değerlerin karşılaştırılmasına dayalı sonuçlarla sınırlı kalmamıştır.

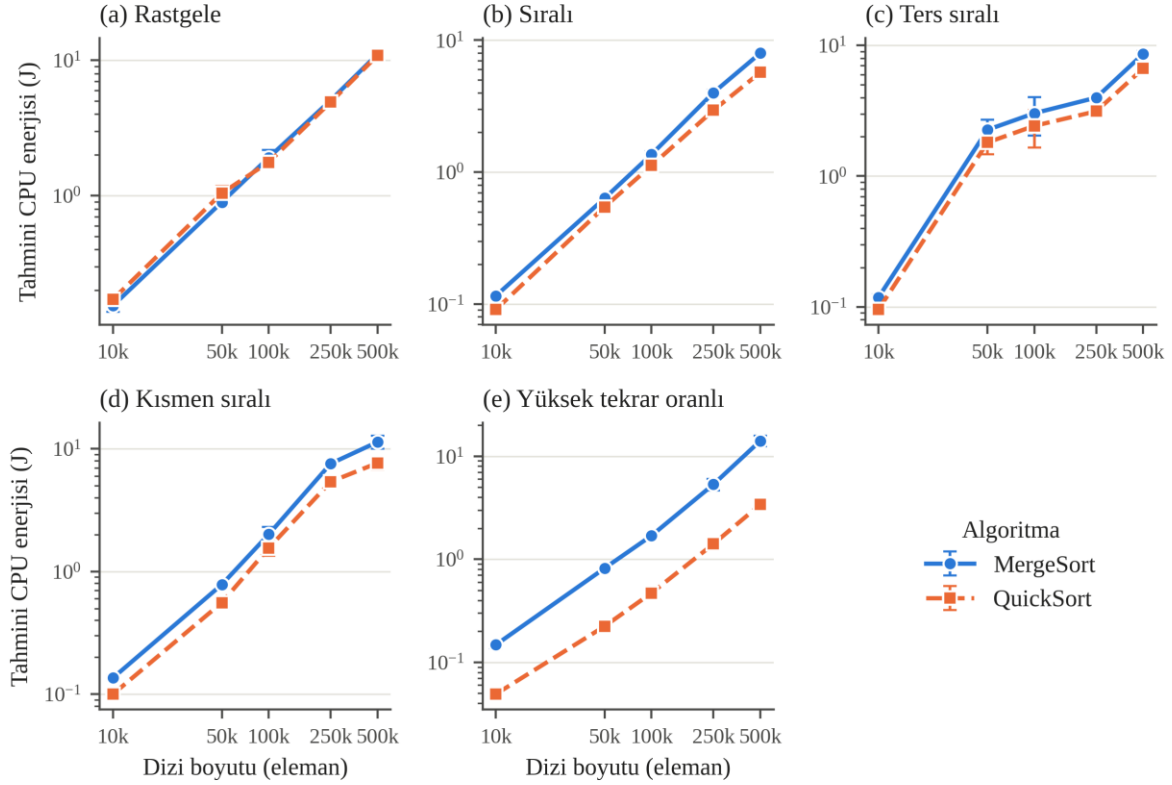
3. BULGULAR VE TARTIŞMA

Bu bölümde nihai deneylerden elde edilen sonuçlar sunulmakta ve tartışılmaktadır. Tablolardaki değerler 20 bağımsız tekrarın ortalaması $\pm$ standart sapması (SS) olarak verilmiş, şekillerdeki hata çubukları ise %95 güven aralığını göstermektedir. Enerji değerleri doğrudan ölçüm değil, CodeCarbon tarafından Windows ortamında CPU yüküne dayalı bir modelle tahmin edilen işlemci enerjisinin (ayrıntılar için Bölüm 3.4) tek bir yürütmeye düşen payı olup okunabilirlik için joule (J) cinsinden verilmiştir ($1 \text{ kWh} = 3,6 \times 10^6 \text{ J}$). Karbon emisyonu, enerji değerinin sabit bir emisyon katsayısıyla ($0,469 \text{ kgCO}_2\text{e/kWh}$) çarpılmasıyla hesaplandığından algoritmalar arasındaki göreceli emisyon farkları enerji farklarıyla aynıdır. Algoritma çiftleri, aynı tohum değeriyle üretilmiş aynı girdiler üzerinde eşleştirilmiş Wilcoxon işaretli sıralar testiyle karşılaştırılmış; p değerleri her ölçüt ailesi içinde Holm yöntemiyle düzeltilmiş ve etki büyüklüğü eşleştirilmiş rank-biserial korelasyon (r) ile raporlanmıştır. Negatif r değerleri, ikinci algoritmanın (QuickSort veya Strassen) daha düşük değer ürettiğini göstermektedir.

3.1. Sıralama Algoritmalarının Sonuçları

Şekil 1 ve Tablo 2, sıralama algoritmalarının beş veri tipi ve beş dizi boyutundaki tahmini işlemci enerji tüketimini göstermektedir. Her iki algortmada da enerji tüketimi dizi boyutuyla birlikte artmış; 500.000 elemanlı dizilerde tek bir sıralama işleminin enerjisi 3,39 J ile 14,09 J arasında tahmin edilmiştir.

Rastgele veri kümelerinde iki algoritma arasında tutarlı bir fark gözlenmemiştir. Beş boyutun dördünde Holm düzeltmesi sonrasında anlamlı fark bulunmamış ($p \geq 0,107$); yalnızca 50.000 elemanlı dizilerde MergeSort lehine anlamlı bir fark belirlenmiştir (QuickSort'ta %16,4 daha yüksek enerji; $p = 0,010$, $r = +0,75$). 100.000 ve üzerindeki boyutlarda ortalamalar arasındaki fark %2,1–7,5 düzeyinde kalmış ve istatistiksel olarak anlamlı bulunmamıştır.



Şekil 1. Sıralama algoritmalarının veri tipi ve dizi boyutuna göre tahmini işlemci enerji tüketimi (yürütme başına, logaritmik eksenler). Hata çubukları %95 güven aralığını göstermektedir ($n=20$)

Sıralı, ters sıralı ve kısmen sıralı veri kümelerinde QuickSort, tüm boyutlarda MergeSort'a göre anlamlı düzeyde daha az enerji tüketmiştir ($p \leq 0,010$; $|r| \geq 0,76$). Bu veri tiplerinde enerji farkı %14,0 ile %32,6 arasında değişmiştir. En büyük fark yüksek tekrar oranlı veride elde edilmiş; QuickSort bu veri tipinde tüm boyutlarda MergeSort'a göre %66,7–75,9 daha az enerji tüketmiştir ($p < 0,001$). Bu koşulda 500.000 elemanlı dizi için sıralama başına emisyon MergeSort'ta 1,84 mg, QuickSort'ta 0,44 mg CO₂e olarak hesaplanmıştır.

Rastgele veri dışındaki koşullarda çalışma süresi sonuçları (Şekil 2, Tablo 3) enerji sonuçlarıyla aynı örneği izlemektedir. Rastgele veride ise fark %2,7–7,0 aralığında kalmış ve yönü boyuta göre değişmiştir: 10.000 elemanda MergeSort, 500.000 elemanda QuickSort anlamlı düzeyde daha hızlıdır; diğer üç boyutta fark anlamlı değildir. Diğer dört veri tipinde QuickSort tüm boyutlarda anlamlı düzeyde daha kısa sürede tamamlanmış; süre kazancı sıralı, ters sıralı ve kısmen sıralı veride %17,7–29,4, yüksek tekrar oranlı veride ise %65,2–76,7 olmuştur.

Bazı koşullarda tekrarlar arası değişkenlik belirgin biçimde yüksektir; örneğin ters sıralı 50.000 ve 100.000 elemanlı dizilerde MergeSort enerjisinin değişim katsayısı sırasıyla %42 ve %69 olarak hesaplanmıştır. Bu koşullarda ölçeklenme eğrisinde düzensizlik görülmekle birlikte (Şekil 1c), aynı girdiler üzerinde yapılan eşleştirilmiş karşılaştırmalar anlamlı ve aynı yönde sonuç vermiştir.

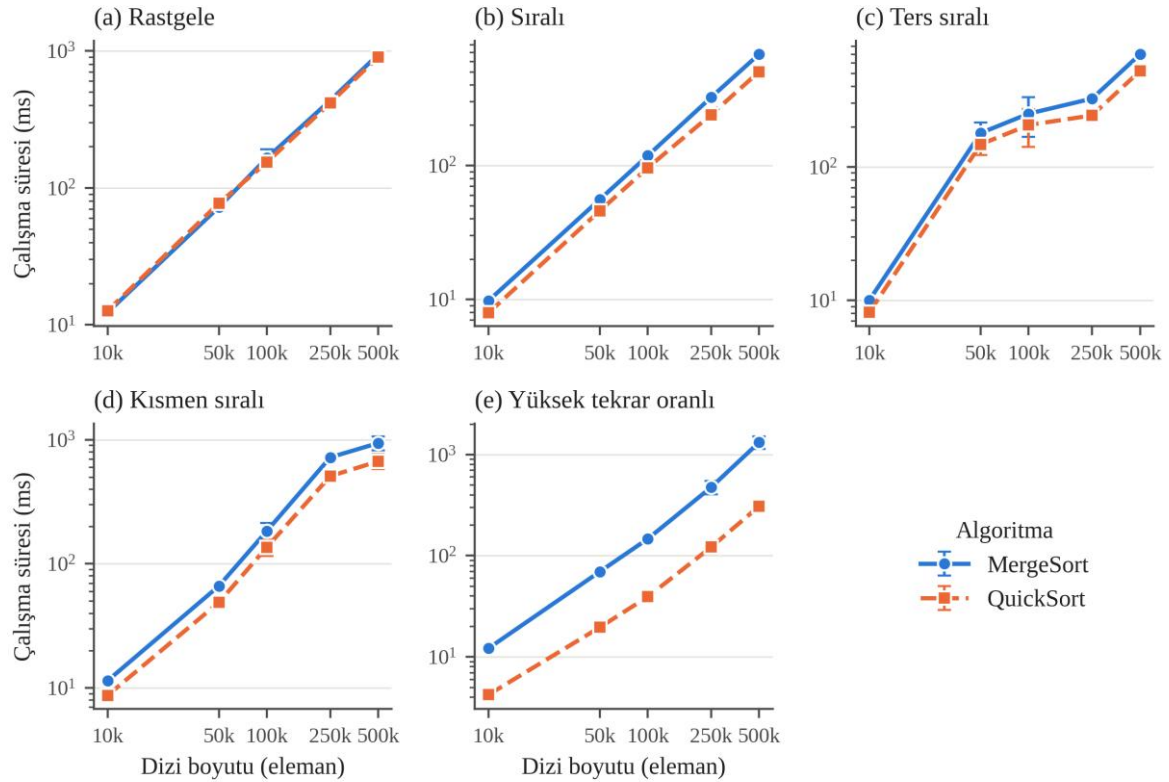
3.2. Matris Çarpımı Algoritmalarının Sonuçları

Şekil 3 ve Tablo 4'te klasik matris çarpımı ile hibrit Strassen algoritmasının sonuçları verilmiştir. Test edilen tüm boyutlarda Strassen uygulaması hem daha kısa çalışma süresi hem de daha düşük tahmini işlemci enerjisi sergilemiş ve farklar Holm düzeltmesi sonrasında anlamlı bulunmuştur ($p \leq 0,009$). Enerji farkı matris boyutuyla birlikte büyümüş; 64×64'te %12,7 olan fark 128×128'de %26,6, 256×256'da %30,3 ve 512×512'de %47,4 düzeyine ulaşmıştır. 512×512 boyutunda bir çarpma işlemi klasik yöntemle ortalama $56,7 \pm 2,5$ s sürmüş ve 466 ± 24 J enerji tüketmiş; Strassen ile bu değerler $28,7 \pm 0,2$ s ve 245 ± 13 J olmuştur. Bu enerji değerleri çarpma başına sırasıyla 60,7 mg ve 31,9 mg CO₂e emisyonu karşılık gelmektedir.

Tablo 2. Sıralama algoritmalarının tahmini işlemci enerji tüketimi (yürütme başına) ve eşleştirilmiş karşılaştırma sonuçları

Veri tipi	Boyut	MergeSort (J)	QuickSort (J)	Fark (%)	<i>p</i> (Holm)	<i>r</i>
Rastgele	10.000	0,154 ± 0,033	0,172 ± 0,037	+11,3	0,107	+0,56
	50.000	0,897 ± 0,121	1,04 ± 0,30	+16,4	0,010*	+0,75
	100.000	1,91 ± 0,58	1,77 ± 0,17	-7,5	1,000	-0,08
	250.000	5,06 ± 0,76	4,91 ± 0,44	-3,1	1,000	-0,21
	500.000	11,1 ± 1,2	10,9 ± 1,0	-2,1	1,000	-0,20
Sıralı	10.000	0,115 ± 0,010	0,0907 ± 0,0073	-21,3	< 0,001*	-0,99
	50.000	0,633 ± 0,069	0,544 ± 0,060	-14,0	< 0,001*	-0,91
	100.000	1,36 ± 0,17	1,12 ± 0,07	-17,7	< 0,001*	-1,00
	250.000	4,00 ± 0,44	2,94 ± 0,27	-26,5	< 0,001*	-1,00
	500.000	7,98 ± 0,55	5,70 ± 0,43	-28,6	< 0,001*	-1,00
Ters sıralı	10.000	0,118 ± 0,012	0,0957 ± 0,0086	-19,2	< 0,001*	-0,92
	50.000	2,26 ± 0,95	1,81 ± 0,72	-20,0	0,010*	-0,76
	100.000	3,03 ± 2,10	2,42 ± 1,63	-20,0	0,005*	-0,81
	250.000	3,99 ± 0,37	3,13 ± 0,25	-21,4	< 0,001*	-1,00
	500.000	8,61 ± 0,88	6,65 ± 0,49	-22,7	< 0,001*	-1,00
Kısmen sıralı	10.000	0,136 ± 0,013	0,1000 ± 0,0088	-26,4	< 0,001*	-0,99
	50.000	0,782 ± 0,077	0,557 ± 0,054	-28,8	< 0,001*	-1,00
	100.000	2,00 ± 0,63	1,55 ± 0,46	-22,6	< 0,001*	-0,90
	250.000	7,56 ± 0,37	5,38 ± 0,35	-28,8	< 0,001*	-1,00
	500.000	11,4 ± 2,9	7,65 ± 1,94	-32,6	< 0,001*	-1,00
Yüksek tekrar oranlı	10.000	0,148 ± 0,020	0,0491 ± 0,0051	-66,7	< 0,001*	-1,00
	50.000	0,819 ± 0,070	0,224 ± 0,016	-72,7	< 0,001*	-1,00
	100.000	1,69 ± 0,12	0,470 ± 0,057	-72,2	< 0,001*	-1,00
	250.000	5,32 ± 1,46	1,41 ± 0,34	-73,6	< 0,001*	-1,00
	500.000	14,1 ± 3,5	3,39 ± 0,71	-75,9	< 0,001*	-1,00

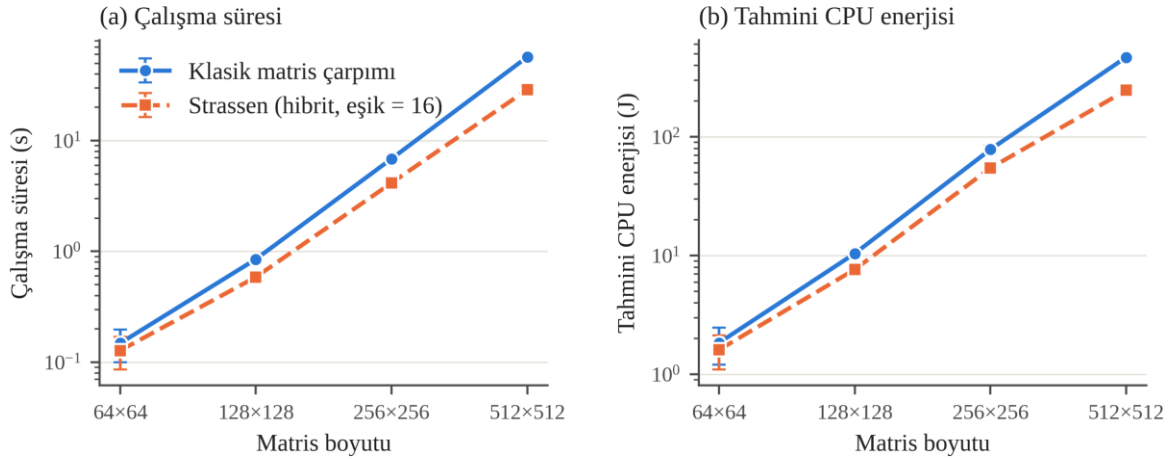
Değerler ortalama ± SS ($n=20$). Fark = (QuickSort - MergeSort) / MergeSort × 100. *p*: eşleştirilmiş Wilcoxon işaretli sıralar testi, Holm düzeltmeli; *r*: eşleştirilmiş rank-biserial korelasyon (negatif değer QuickSort lehinedir). * $p < 0,05$.

**Şekil 2.** Sıralama algoritmalarının veri tipi ve dizi boyutuna göre çalışma süresi (yürütme başına, logaritmik eksenler). Hata çubukları %95 güven aralığını göstermektedir ($n=20$)

Tablo 3. Sıralama algoritmalarının çalışma süresi (yürütme başına) ve eşleştirilmiş karşılaştırma sonuçları

Veri tipi	Boyut	MergeSort (ms)	QuickSort (ms)	Fark (%)	p (Holm)	r
Rastgele	10.000	12,3 ± 0,3	12,6 ± 0,3	+2,7	< 0,001*	+0,95
	50.000	72,2 ± 1,1	77,1 ± 19,2	+6,8	1,000	+0,11
	100.000	165 ± 55	153 ± 3	-7,0	1,000	+0,10
	250.000	434 ± 58	416 ± 8	-4,2	0,072	-0,57
	500.000	938 ± 21	900 ± 28	-4,0	< 0,001*	-0,94
Sıralı	10.000	9,74 ± 0,07	7,91 ± 0,07	-18,8	< 0,001*	-1,00
	50.000	55,7 ± 0,4	45,8 ± 0,3	-17,8	< 0,001*	-1,00
	100.000	118 ± 2	96,0 ± 0,9	-18,7	< 0,001*	-1,00
	250.000	322 ± 6	239 ± 5	-25,8	< 0,001*	-1,00
	500.000	678 ± 6	501 ± 5	-26,0	< 0,001*	-1,00
Ters sıralı	10.000	10,0 ± 0,1	8,11 ± 0,22	-19,1	< 0,001*	-1,00
	50.000	179 ± 75	147 ± 53	-17,7	0,008*	-0,75
	100.000	250 ± 177	206 ± 138	-17,7	0,004*	-0,81
	250.000	324 ± 3	242 ± 3	-25,2	< 0,001*	-1,00
	500.000	696 ± 11	522 ± 12	-25,0	< 0,001*	-1,00
Kısmen sıralı	10.000	11,4 ± 0,2	8,70 ± 0,18	-23,7	< 0,001*	-1,00
	50.000	66,2 ± 1,6	48,8 ± 0,9	-26,3	< 0,001*	-1,00
	100.000	184 ± 63	136 ± 44	-26,0	0,001*	-0,88
	250.000	719 ± 27	508 ± 18	-29,4	< 0,001*	-1,00
	500.000	937 ± 255	669 ± 195	-28,6	< 0,001*	-1,00
Yüksek tekrar oranlı	10.000	12,1 ± 0,2	4,21 ± 0,23	-65,2	< 0,001*	-1,00
	50.000	69,4 ± 1,2	19,5 ± 1,0	-71,8	< 0,001*	-1,00
	100.000	146 ± 3	39,1 ± 2,0	-73,3	< 0,001*	-1,00
	250.000	473 ± 152	121 ± 36	-74,4	< 0,001*	-1,00
	500.000	1317 ± 387	307 ± 75	-76,7	< 0,001*	-1,00

Değerler ortalama ± SS ($n=20$). Fark = (QuickSort - MergeSort) / MergeSort × 100. p : eşleştirilmiş Wilcoxon işaretli sıralar testi, Holm düzeltmeli; r : eşleştirilmiş rank-biserial korelasyon (negatif değer QuickSort lehinedir). Fark ortalamalara, r ise eşleştirilmiş farkların sıralarına dayandığından aykırı değer içeren koşullarda (ör. rastgele, 100.000) işaretleri farklı olabilir. * $p < 0,05$.



Şekil 3. Klasik matris çarpımı ve hibrit Strassen algoritmasının (a) çalışma süresi ve (b) tahmini işlemci enerji tüketimi (logaritmik eksenler). Hata çubukları %95 güven aralığını göstermektedir ($n=20$)

64×64 boyutunda tekrarlar arası değişkenlik çok yüksektir (enerji için değişim katsayısı %68–74) ve etki büyüklüğü daha küçüktür ($r = -0,65$); bu nedenle en küçük boyuttaki fark dikkatle yorumlanmalıdır. 128×128 ve üzerindeki boyutlarda değişim katsayısı en fazla %19 olmuş ve etki büyüklüğü $|r| \geq 0,97$ olmuştur.

Tablo 4. Matris çarpımı algoritmalarının çalışma süresi, tahmini işlemci enerji tüketimi, karbon emisyonu ve eşleştirilmiş karşılaştırma sonuçları

Boyut	Yöntem	Süre (s)	Tahmini enerji (J)	CO ₂ e (mg)	Enerji farkı (%)	<i>p</i> süre	<i>p</i> enerji	<i>r</i> enerji
64×64	Klasik	0,148 ± 0,103	1,84 ± 1,36	0,239 ± 0,178	–	–	–	–
	Strassen	0,127 ± 0,089	1,60 ± 1,09	0,209 ± 0,142	–12,7	< 0,001*	0,009*	–0,65
128×128	Klasik	0,848 ± 0,008	10,4 ± 1,3	1,35 ± 0,17	–	–	–	–
	Strassen	0,585 ± 0,005	7,61 ± 1,08	0,992 ± 0,141	–26,6	< 0,001*	< 0,001*	–0,97
256×256	Klasik	6,84 ± 0,09	78,0 ± 12,4	10,2 ± 1,6	–	–	–	–
	Strassen	4,14 ± 0,05	54,4 ± 10,2	7,08 ± 1,33	–30,3	< 0,001*	< 0,001*	–1,00
512×512	Klasik	56,7 ± 2,5	466 ± 24	60,7 ± 3,1	–	–	–	–
	Strassen	28,7 ± 0,2	245 ± 13	31,9 ± 1,7	–47,4	< 0,001*	< 0,001*	–1,00

Değerler ortalama ± SS (*n*=20). CO₂e = enerji × 0,469 kgCO₂e/kWh. Fark, *p* ve *r* değerleri Strassen satırında klasik yöntemle karşılaştırılmıştır. *p*: eşleştirilmiş Wilcoxon işaretli sıralar testi, Holm düzeltmeli; *r*: eşleştirilmiş rank-biserial korelasyon (negatif değer Strassen lehinedir). * *p* < 0,05.

3.3. Tartışma

Genel olarak tahmini enerji tüketimi ile çalışma süresi birlikte değişmiş; daha kısa sürede tamamlanan algoritma genellikle daha az enerji tüketmiştir. Bu sonuç yorumlanırken kullanılan tahmin yönteminin niteliği göz önünde bulundurulmalıdır: CodeCarbon işlemci gücünü CPU yüküne dayalı bir modelle tahmin ettiğinden, aynı yük düzeyinde çalışan tek iş parçacıklı iki algoritma için tahmini güç birbirine yakın olmakta ve enerji farkları büyük ölçüde çalışma süresi farklarını yansıtmaktadır. Dolayısıyla bu çalışmadaki enerji karşılaştırmaları, algoritmaların donanım düzeyindeki güç davranışındaki farklardan çok, yürütme süresi farklarının tahmini enerjiye yansımaları göstermektedir. Bu bulgu, işlemciye bağlı iş yüklerinde algoritmalar arasındaki enerji farklarının büyük ölçüde yürütme süresindeki farklardan kaynaklandığını bildiren çalışmalarla uyumludur [5,13]. Yazılım düzeyindeki değişikliklerin enerji tüketimine etkisinin sistematik ve tekrarlı ölçümlerle incelenmesi gerektiğini vurgulayan çalışmalar da bu çalışmadaki eşleştirilmiş ve tekrarlı deney tasarımı desteklemektedir [14,15].

Sayısal doğrusal cebir uygulamalarında teorik işlem sayısı kadar özyineleme maliyeti, alt matrislerin oluşturulma biçimi ve taban durum eşiği gibi uygulama ayrıntıları da gerçek performansı belirlemektedir [16]. Strassen algoritması klasik yöntemin $O(n^3)$ olan çarpma sayısını yaklaşık $O(n^{2,81})$ düzeyine indirmektedir [11]; ancak bu asimptotik avantajın pratikte ortaya çıkıp çıkmadığı söz konusu uygulama ayrıntılarına bağlıdır. Kullanılan hibrit uygulamada 16×16 ve daha küçük alt problemler klasik yöntemle çözülmektedir. Bu durumda $n \times n$ boyutlu bir çarpmada $k = \log_2(n/16)$ özyineleme düzeyi oluşur ve skaler çarpma sayısı klasik yöntemle göre $(7/8)^k$ oranına iner; bu oran 64, 128, 256 ve 512 boyutları için sırasıyla 0,77, 0,67, 0,59 ve 0,51 değerlerini almaktadır. Ölçülen çalışma süresi oranları (0,86, 0,69, 0,61 ve 0,51) bu teorik değerlere yakındır ve aradaki sapma boyut arttıkça küçülmektedir. Bu gözlem, test edilen koşullarda Strassen'in avantajının azalan skaler işlem sayısı ile tutarlı olduğunu göstermektedir ve Strassen algoritmasının uygun uygulama tercihleriyle verimlilik sağlayabileceğini bildiren çalışmalarla uyumludur [6,7]. Bununla birlikte Strassen uygulamasındaki alt matris toplama ve çıkarma işlemlerinin NumPy vektör işlemleriyle, klasik yöntemdeki skaler çarpımların ise saf Python döngüleriyle yapıldığı göz önünde bulundurulmalıdır; bu uygulama farkı Strassen lehine ek bir avantaj oluşturmuş olabilir.

Bu bulgular, bir algoritmanın enerji davranışının yalnızca adıyla veya asimptotik karmaşıklığıyla belirlenmediğini, veri yapısı ve uygulama biçimi gibi tercihlerin de sonucu doğrudan etkileyebileceğini göstermektedir. Uygulama geliştiricilerin enerji verimliliği için bu tür tercihleri dikkate alması gerektiği literatürde de vurgulanmaktadır [17,18]. Bu açıdan sonuçlar, enerji ve kaynak verimliliğinin yazılım yaşam döngüsünde bir kalite ölçütü olarak ele alınmasını öneren sürdürülebilir yazılım yaklaşımlarıyla uyumludur [19–22].

Bu çalışmada önbellek isabet oranı, bellek erişim sayısı veya bellek tahsisi doğrudan ölçülmemiştir. Bellek hiyerarşisinin performans ve enerji üzerinde önemli bir etkisi olduğu bilinmekle birlikte [23], gözlenen farkların bu etkenlerden hangisine ne ölçüde bağlı olduğu mevcut verilerle belirlenmemektedir. Bu nedenle bellek ve önbellek davranışına dayanan açıklamalar kesin bulgu olarak değil, sonraki çalışmalarda sınanması gereken olası etkenler olarak değerlendirilmelidir.

Rastgele veride iki sıralama algoritması arasında tutarlı bir fark gözlenmemesi, her iki algoritmanın bu veri tipinde ortalama $O(n \log n)$ karşılaştırma yapmasıyla uyumludur. 50.000 elemanlı koşulda gözlenen tek anlamlı farkın yönünün 100.000 ve üzerindeki boyutlardaki eğilimle tutarlı olmaması, küçük farkların ölçüm değişkenliğine duyarlı olduğunu göstermektedir. Bu nedenle rastgele veri için iki algoritmadan birinin enerji açısından üstün olduğu sonucuna varılmamıştır.

Kullanılan QuickSort uygulaması pivotu alt dizinin ortasından seçtiği için, ilk veya son elemanı pivot alan uygulamalarda sıralı ve ters sıralı girdilerde görülen $O(n^2)$ en kötü durum bu girdilerde oluşmamıştır. Sıralı, ters sıralı ve kısmen sıralı verilerde QuickSort'un MergeSort'a göre gözlenen avantajının kaynağı ise bu çalışmada ayrıca incelenmemiştir. Yüksek tekrar oranlı veride gözlenen büyük fark ise üç yollu bölümlenmeyle açıklanabilir: pivota eşit elemanlar tek adımda ayrılıp özyinelemeye dahil edilmediğinden, tekrar oranı arttıkça işlenmesi gereken alt problemlerin boyutu küçülmektedir [24]. İki yollu bölümlenme kullanan bir QuickSort uygulamasında aynı avantajın görülmesi beklenmediğinden bu sonuç kullanılan uygulamaya özgü olarak yorumlanmalıdır. Ayrıca QuickSort uygulamasında bölümlenme adımının Python liste üreteçleriyle, MergeSort uygulamasında ise birleştirme adımının açık döngülerle yapıldığı dikkate alınmalıdır; bu yapıların yorumlayıcı maliyetleri arasındaki farkın gözlenen sonuçlara katkısı bu çalışmada ayrıca ölçülmemiştir.

3.4. Sınırlılıklar

Bu çalışmanın bulguları bazı sınırlılıklar çerçevesinde değerlendirilmelidir. İlk olarak, tüm deneyler tek bir dizüstü bilgisayarda (Intel Core i7-13700H, Windows 11) ve tek bir Python sürümüyle (3.11.9) yürütülmüştür; farklı işlemci mimarilerinde, işletim sistemlerinde, güç yönetimi ayarlarında veya Python sürümlerinde sonuçlar farklılaşabilir. Deneyler Windows'un dengeli güç modunda, işlemci frekansı ve turbo özelliği sabitlenmeden yürütülmüş; işlemci sıcaklığı ölçüm sırasında izlenmemiştir. Isınma çalışmaları, dönüşümlü çalışma sırası ve eşleştirilmiş tasarım bu etkileri azaltmakla birlikte tamamen ortadan kaldırmamaktadır. İkinci olarak, algoritmalar saf Python ile gerçekleştirilmiştir ve derlenen dillerde ya da optimize edilmiş kütüphanelerde farklı sonuçlar elde edilebilir. Özellikle klasik matris çarpımı üç döngülü saf Python uygulamasıdır; Strassen algoritmasının klasik yöntemle üstünlüğünün yüksek başarımlı optimize uygulamalarla yapılan karşılaştırmalarda farklılaşabildiği bilinmektedir [25].

Üçüncü olarak, enerji tüketimi donanım tabanlı güç ölçerlerle değil, yazılım tabanlı bir araç olan CodeCarbon ile tahmin edilmiştir. Windows ortamında Intel RAPL sayaçlarına doğrudan erişilemediğinden CodeCarbon işlemci enerjisini CPU yüküne dayalı bir modelle hesaplamıştır. Bu tür araçların doğruluğu platforma ve kullanılan ölçüm yöntemine göre değişmektedir [8,26]. Ayrıca işlemci modeli CodeCarbon'un güç veri tabanında yer almadığından araç, iş parçacığı başına 4 W varsayılan değerini (20 iş parçacığı için 80 W) kullanmıştır; bu değer işlemcinin üretici tarafından belirtilen 45 W temel gücünden farklıdır [27]. Bu varsayımın tüm tahminlere aynı ölçekte uygulanması nedeniyle algoritmalar arasındaki görece farkları etkilemeyeceği varsayılmıştır; ancak raporlanan mutlak enerji ve emisyon değerleri bu varsayıma bağlıdır ve algoritmaların donanım düzeyindeki güç tüketimi farkları doğrudan gözlemlenmemektedir. Bu nedenle çalışmanın bulguları mutlak değerlerden çok, aynı koşullarda elde edilen algoritmalar arası görece farklara dayanmaktadır. Toplu ölçüm ve eşleştirilmiş tasarım ölçüm gürültüsünü azaltmakla birlikte, 64×64 matris ve bazı sıralama koşullarında tekrarlar arası değişkenliğin yüksek kaldığı görülmüştür. Ayrıca 128×128 ve 256×256 matris koşullarında çalışma süresinin değişim katsayısı yaklaşık %1 iken tahmini enerjinin değişim katsayısı yaklaşık %13–19 olmuştur. CPU yüküne dayalı model makine genelindeki işlemci yükünü kullandığından ve deneyler sırasında sistem yükü ayrıca sabitlenmediğinden, arka plan yükünün bu ek değişkenliğe katkıda bulunmuş olabileceği değerlendirilmektedir.

Son olarak, deneyler seçili girdi boyutlarıyla (sıralama için 10.000–500.000 eleman, matris çarpımı için en fazla 512×512) sınırlıdır. 1024×1024 boyutu, saf Python klasik çarpımında $O(n^3)$ ölçeklenmeye göre tek bir işlemin yaklaşık 7,6 dakika süreceği öngörüldüğünden 20 tekrarlı tasarıma dahil edilmemiştir; tüm matris boyutlarında aynı ölçüm protokolünü korumak amacıyla nihai matris seti 64 – 512 aralığıyla sınırlandırılmıştır. Bölüm 3.3'teki $(7/8)^k$ ilişkisine göre 1024×1024 boyutunda teorik oran 0,45 olup farkın aynı yönde büyümesi beklenmektedir; ancak bu beklenti deneysel olarak doğrulanmamıştır. Bu nedenle elde edilen sonuçlar test edilen koşullar ve değerlendirilen uygulamalarla sınırlıdır.

4. SONUÇLAR

Bu çalışmada böl ve yönet yaklaşımına dayalı iki sıralama ve iki matris çarpımı algoritması, saf Python uygulamaları üzerinden çalışma süresi, tahmini işlemci enerjisi ve karbon emisyonu açısından 20 tekrarlı eşleştirilmiş bir deney tasarımıyla karşılaştırılmıştır.

Test edilen koşullarda elde edilen başlıca sonuçlar şunlardır: (i) rastgele veride MergeSort ile QuickSort arasında tutarlı ve anlamlı bir enerji farkı bulunmamıştır; (ii) sıralı, ters sıralı ve kısmen sıralı veride QuickSort uygulaması %14–33, yüksek tekrar oranlı veride ise üç yollu bölümlenmeyle ilişkili olarak %67–76 daha az enerji tüketmiştir; (iii) eşik değeri 16 olan hibrit Strassen uygulaması, 64×64 – 512×512 aralığındaki tüm boyutlarda saf Python klasik çarpıma göre daha az enerji tüketmiş ve bu fark 512×512 boyutunda %47 düzeyine ulaşmıştır; (iv) karbon emisyonu enerjiden sabit bir katsayıyla hesaplandığından algoritmalar arasındaki göreceli emisyon farkları enerji farklarıyla aynıdır.

Bu sonuçlar, bir algoritmanın enerji davranışının yalnızca adıyla veya asimptotik karmaşıklığıyla değil; veri dağılımı ile pivot seçimi, bölümlenme yöntemi ve taban durum eşiği gibi uygulama tercihleriyle birlikte değerlendirilmesi gerektiğini göstermektedir. Bulgular test edilen donanım, Python sürümü ve uygulamalarla sınırlı olup genellenebilmeleri için farklı platformlarda doğrulanmaları gerekmektedir. Ayrıca enerji değerleri CPU yüküne dayalı bir modelle tahmin edildiğinden, algoritmalar arasındaki enerji farkları büyük ölçüde çalışma süresi farklarını yansıtmaktadır.

Gelecek çalışmalarda deneylerin farklı işlemci mimarilerinde, derlenen dillerde ve optimize edilmiş kütüphanelerle tekrarlanması, enerji tahminlerinin donanım tabanlı güç ölçerlerle doğrulanması ve önbellek ile bellek erişimine ilişkin donanım sayaçlarının ölçüme dahil edilmesi planlanmaktadır.

ETİK KURUL ONAYI VE ÇIKAR ÇATIŞMASI BEYANI

Hazırlanan makalede etik kurul izni alınmasına gerek yoktur.

Hazırlanan makalede herhangi bir kişi/kurum ile çıkar çatışması bulunmamaktadır.

ÜRETKEN YAPAY ZEKA BEYANI

Bu çalışmanın hazırlanmasında üretken yapay zeka araçlarından (Büyük Dil Modelleri), literatür taraması sürecinde kaynak tespiti, metin akıcılığının iyileştirilmesi ve İngilizce çeviri kontrolü amacıyla destek alınmıştır. Çalışmanın deneysel tasarımı, veri toplama, kod geliştirme, analiz ve sonuç çıkarımları yazarlar tarafından gerçekleştirilmiştir. Üretken yapay zeka araçları tarafından üretilen tüm metinler yazarlar tarafından gözden geçirilmiş ve doğruluğu onaylanmıştır.

KAYNAKLAR

- [1] Oukil K, Chiba Z, Moussaid K, Abghour N. Green cloud computing: A literature review on the current state of research. Proc ISNCC 2024. doi:10.1109/ISNCC62547.2024.10758953.
- [2] Gupta S, Gupta A. Advancing carbon-efficient software development: A sustainable path forward. Proc ICDCN 2025;1–6. doi:10.1145/3700838.3703670.
- [3] Vankudre A, Rani V. Energy efficient algorithms in green cloud computing: A comprehensive study. Proc ICCCN 2023. doi:10.1109/ICCCN56998.2023.10306976.
- [4] Camargo-Henríquez I, Martínez-Rojas A. Energy optimization in software: A comparative analysis of programming languages and code-execution strategies. Proc IESTEC 2024. doi:10.1109/IESTEC62784.2024.10820312.
- [5] Schmitt N, Kamthania S, Rawtani N, Mendoza L, Lange KD, Kounev S. Energy-efficiency comparison of common sorting algorithms. Proc MASCOTS 2021. doi:10.1109/MASCOTS53633.2021.9614299.
- [6] Oo NZ, Chaikan P. The effect of loop unrolling in energy efficient Strassen's algorithm on shared memory architecture. Proc ITC-CSCC 2021. doi:10.1109/ITC-CSCC52171.2021.9501472.
- [7] Krishnan AG, Goswami D. Multi-stage memory efficient Strassen's matrix multiplication on GPU. Proc HiPC 2021. doi:10.1109/HIPC53243.2021.00035.

- [8] Raffin G, Trystram D. Dissecting the software-based measurement of CPU energy consumption: A comparative analysis. *IEEE Trans Parallel Distrib Syst* 2025;36(1). doi:10.1109/TPDS.2024.3492336.
- [9] El Ghabzouri M, Es Salhi A. Python overheads using Ackermann function benchmark: Performance and power consumption. *Proc ISEC* 2025. doi:10.1109/ISEC64801.2025.11147277.
- [10] Cheng W, Hu W, Ma X. Revisiting knowledge-based inference of Python runtime environments: A realistic and adaptive approach. *IEEE Trans Softw Eng* 2024;50(2):258–279. doi:10.1109/TSE.2023.3346474.
- [11] Strassen V. Gaussian elimination is not optimal. *Numer Math* 1969;13(4):354–356. doi:10.1007/BF02165411.
- [12] T.C. Enerji ve Tabii Kaynaklar Bakanlığı. Türkiye elektrik üretimi ve elektrik tüketim noktası emisyon faktörleri bilgi formu (2023 yılı verisi). Ankara: ETKB.
- [13] Seo M, Dang VN, Kim H. Performance and energy consumption analysis of sorting algorithms. *J Syst Archit* 2021;117:102145. doi:10.1016/j.sysarc.2021.102145.
- [14] Hindle A. Green mining: A methodology of relating software change and configuration to power consumption. *Empir Softw Eng* 2015;20(2):374–409. doi:10.1007/s10664-013-9276-6.
- [15] Sahin S, Tosun A, Tekinerdogan B, Aksit M. Energy efficiency in software: A systematic literature review. *ACM Comput Surv* 2021;54(5):1–35. doi:10.1145/3447588.
- [16] Demmel J. *Applied Numerical Linear Algebra*. Philadelphia: SIAM; 1997.
- [17] Pinto G, Castor F. Energy efficiency: A new concern for application software developers. *Commun ACM* 2017;60(12):68–75. doi:10.1145/3125374.
- [18] Cruz L, Abreu R. Catalog of energy patterns for mobile applications. *Empir Softw Eng* 2019;24(4):2209–2235. doi:10.1007/s10664-018-9661-9.
- [19] Lago P, Bezemer C, de Souza F, Guéhéneuc T. Sustainability and software engineering: Trends and future directions. *Proc ICSE Companion* 2017;487–488. doi:10.1109/ICSE-C.2017.159.
- [20] Kern E, Naumann S, Dick M, Johann T, Lucas N. Sustainable software products—Towards assessment criteria for resource and energy efficiency. *Future Gener Comput Syst* 2018;86:199–210. doi:10.1016/j.future.2018.02.044.
- [21] Yalçın Ö. Ekipmanlarda Ağırlık Hafifletilmesi ile Enerji Tüketimi ve Karbon Ayak İzi Azaltımı. *DEUFMD* 2026;28(83):225–236. doi:10.21205/deufmd.2026288308.
- [22] Naumann S, Dick M, Kern E, Johann T. The GREENSOFT model: A reference model for green and sustainable software and its engineering. *Sustain Comput Inform Syst* 2011;1(4):294–304. doi:10.1016/j.suscom.2011.06.004.
- [23] Hennessy JL, Patterson DA. *Computer architecture: A quantitative approach*. 6th ed. Morgan Kaufmann; 2017.
- [24] Bentley JL, McIlroy MD. Engineering a sort function. *Softw Pract Exp* 1993;23(11):1249–1265. doi:10.1002/spe.4380231105.
- [25] Huang J, Smith TM, Henry GM, van de Geijn RA. Strassen’s algorithm reloaded. *Proc SC16* 2016;690–701. doi:10.1109/SC.2016.58.
- [26] Jay M, Ostapenko V, Lefèvre L, Trystram D, Orgerie AC, Fichel B. An experimental comparison of software-based power meters: Focus on CPU and GPU. *Proc CCGrid* 2023;106–118. doi:10.1109/CCGrid57682.2023.00020.
- [27] Intel Corporation. Intel Core i7-13700H processor (24M cache, up to 5.00 GHz): Product specifications. <https://www.intel.com/content/www/us/en/products/sku/232128/intel-core-i713700h-processor-24m-cache-up-to-5-00-ghz/specifications.html> [Erişim: 29.09.2026].