Paralel Kelime Üretici ve Kelime Oluşturucu Algoritmalar Geliştirilmesi

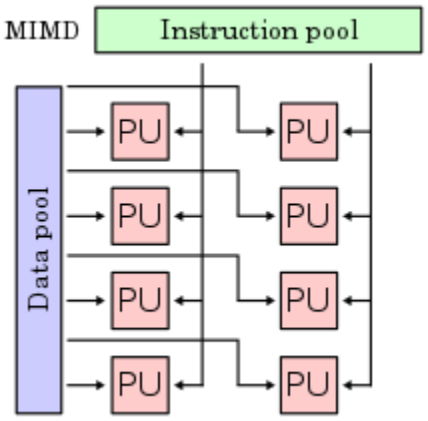
Fatih ATEŞ
Bursa Teknik Üniversitesi, 19360859074@btu.edu.tr

***Özetçe*- Eş zamanlılık aslında günlük hayatımızda da çokça uyguladığımız bir eylemdir. Örneğin bir elimizde su şişesi tutarken diğer elimizle kapıyı açmak gibi. Paralel Programlama da eş zamanlı işlemlerin bilgisayar üzerinde gerçekleştirilmesinden ibarettir. Bildiri içerisinde MPI kütüphanesi kullanılarak bir Kelime Üreticinin geliştirilme süreci ve MapReduce algoritmasını kendi ihtiyaçlarıma göre değiştirerek geliştirdiğim bir Kelime sayıcının geliştirilme yöntemleri anlatılmıştır. Yapılan değerlendirmeler sonucunda da Paralleştirmenin bir noktadan sonra zaman anlamında yarardan çok zarar getirdiği ortaya koyulmuştur. Bu da Amdhal yasasınının geçerliliğinide kanıtlamaktadır.**

Anahtar Kelimeler- MPI, C++, MapReduce, Go, Concurrency, Process, Thread, Go routine

1. GİRİŞ

Eş zamanlılık aslında günlük hayatımızda da çokça uyguladığımız bir eylemdir. Örneğin bir elimizde su şişesi tutarken diğer elimizle kapıyı açmak gibi. Paralel Programlama da eş zamanlı işlemlerin bilgisayar üzerinde gerçekleştirilmesinden ibarettir. Eş zamanlı işlemler gerçekleştirebilmeniz için öncelikle buna uygun mimaride bir CPU edinmeniz gerekmektedir. Günümüzdeki işlemcilerin çoğunluğu eş zamanlı işlemler gerçekleştirebilmektedir. Evinizde kullandığınız bir işlemci Şekil 1.1 üzerinde görülen MIMD mimarisini kullanmaktadır.



Şekil 1.1 MIMD (Wikipedia, 2022)

2. MPI

MPI (Message Passing Interface) bir bilgisayar iletişim protokolüdür. Dağıtık bellekli bir sistemde paralel program koşan düğümlerin arasındaki iletişim için kullanılan fiilen standart bir protokoldür. Hem noktadan noktaya hem de toplu iletişim desteklenir. MPI, özelliklerinin herhangi bir uygulamada nasıl davranması gerektiğine ilişkin protokol ve anlamsal spesifikasyonlarla birlikte bir mesaj ileten uygulama programcısı arayüzüdür. MPI'nin hedefleri yüksek performans, ölçeklenebilirlik ve taşınabilirliktir. MPI, bugün yüksek performanslı bilgi işlemde kullanılan baskın model olmaya devam ediyor. MPI herhangi bir kuruluş tarafından onaylanmamasına rağmen de facto bir standart haline gelmiştir (Wikipedia, 2022).

3. MapReduce

MapReduce, bir küme üzerinde dağıtılmış büyük veri kümelerini işlemek ve oluşturmak için bir programlama modeli ve ilişkili bir uygulamadır. MapReduce veriyi dağıtmak ve sonrasında indirgeyerek geri toplamak için sunulmuştur bir algoritmadır. Aynı ismi taşıyan Google'ın Big Data için sağladığı bir MapReduce çatısı da mevcuttur. Ancak bu bildiri içerisinde bu çatı kullanılmamıştır. Tamamen MapReduce algoritması aracın ihtiyaçlarına göre optimizde edilerek geliştirilmiştir. Şekil 3.1 üzerinde kabaca bir döküman içerisindeki verileri saymak için kullanılan MapReduce algoritmasını görebilirsiniz. Görselde paralel bir işlem kullanılmamıştır. Bu demektir ki MapReduce yalnızca paralel işlemlerde kullanılmamaktadır.

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += pc
  emit (word, sum)
```

Şekil 3.1 MapReduce pseudo kodu (Wikipedia, 2022)

4. KELİME ÜRETİCİ

Kelime üretici verilen “n” sayıda process ile yine kullanıcının belirleyebildiği minimum kelime uzunluğu ve maksimum kelime uzunluğu kısıtlarına göre İngiliz alfabesinde bulunan 26 karakter içerisinden rastgele seçilen karakterlerin birleştirilmesiyle bir kelime oluşturarak bu kelimeleri de aralarında boşluklar olacak şekilde dosyaya kaydeden bir araçtır.

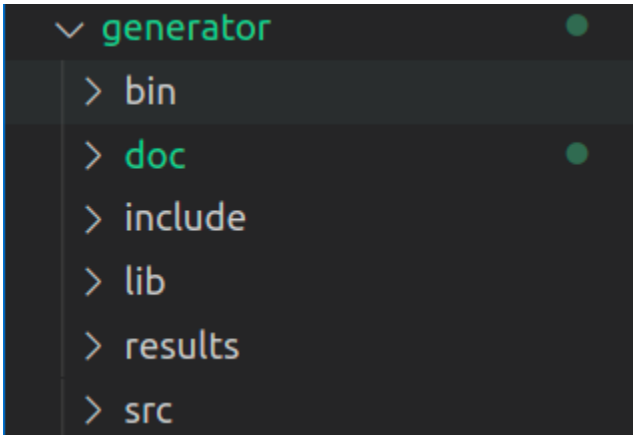
Kelime üreticinin pseudo kodu Şekil 4.1 üzerinde görülebilmektedir.

```
MPI_INIT()
MPI_Comm_rank()
MPI_Comm_size(N)

yeni Generator üret
if world_rank == 0:
   Her processe gönderilecek işlem sayısını hesapla
   Her processten ne kadar veri alınacağını hesapla

scatter için reciver tamponu oluştur

ürettiğin verileri dağıt
bariyerde tüm veriler dağıtılana kadar bekle

kelimeleri tutabilmek için bir char tamponu oluştur
kelimeleri oluştur ve char tamponuna eşitle

if world_rank == 0:
   tüm kelimelerin toplanacağı bir tampon oluştur

tüm processlerden verileri tampona toparla

tüm veriler toplanana kadar bariyerde bekle

if world_rank == 0:
   tüm verileri dosyaya yaz
   çalışma için geçen süreyi ekrana yazdır

MPI_Finalize()

oluşturduğun tamponu serbest bırak
```

Şekil 4.1 Kelime Üretici pseudo kodu

Kelime üretici C++ üzerinde MPI kütüphanesi kullanılarak gerçekleştirilmiştir. Kabaca yapılan işlem istenilen kelime sayısını proccess sayısına bölerek her processe oluşturması gereken kelime sayısı bufferlar aracılığıyla iletilmektedir. Her process kendi payına düşen kelimeleri ürettikten sonra root process içerisinde tüm kelimeler toplanmaktadır. Toplanan kelimeler aralarına birer boşluk yerleştirilerek bir dosyaya kaydedilmektedir.

Kelime üretici aracının dosya yapısı Şekil 4.2 üzerinde görülmektedir.



Şekil 4.2 Kelime üretici dosya yapısı

- /bin dizini altında üretilen çalıştırılabilir dosyalar bulunmaktadır.
- /doc dizini altında gerekli dökümanlar bulunmaktadır.
- /include dizini altında başlık dosyaları bulunmaktadır.
- /lib dizini altında yazılan kaynak kodunun kütüphane halinde derlenmiş C++ başlık dosyaları bulunmaktadır.
- /results dizini altında üretilmiş olan veri setleri mevcuttur.
- /src dizini altında başlık dosyalarının kaynak kodları bulunmaktadır.

5. KELİME SAYICI

Kelime sayıcı verilen “n” sayıda process ile verilen dosyanın içerisindeki kelimelerden kaçar adet bulunduğu saymaktadır. Sayma işlemi tamamlandıktan sonra bu kelimeleri YAML formatında key-value eşleriyle dosyaya yazan bir araçtır.

Kelime sayıcının pseudo kodu Şekil 5.1 üzerinde görülmektedir.

```
process kuyruğu oluştur

function dosyadan_oku:
   buff = 64
   “buff” byte veri oku ve bu veriden kelimeleri ayıkla
   tamamen okunmuş kelimeleri bir hashmape at
   tamamen okunamayan kelimeleri bir diziye at

function map:
   processin ne kadar boyutta veri okuyacağını belirle
   go rutini çalıştır ve içerisinde "dosyadan_oku" metodunu çalıştır
   processi, process kuyruğunu at

function reduce:
   while:
      tüm processlerin kuyruktan çıkmasını bekle

   words = {}
   for p in processes:
      tamamen okunmayan kelimeleri geçici belleğe at ve \
      bir sonraki processten gelecek olanlar ile birleştir

   for p in processes:
      processte tamamen okunmuş verileri words değişkeni içerisine aktar


map()
reduce()
kelimeleri dosyaya kaydet
```

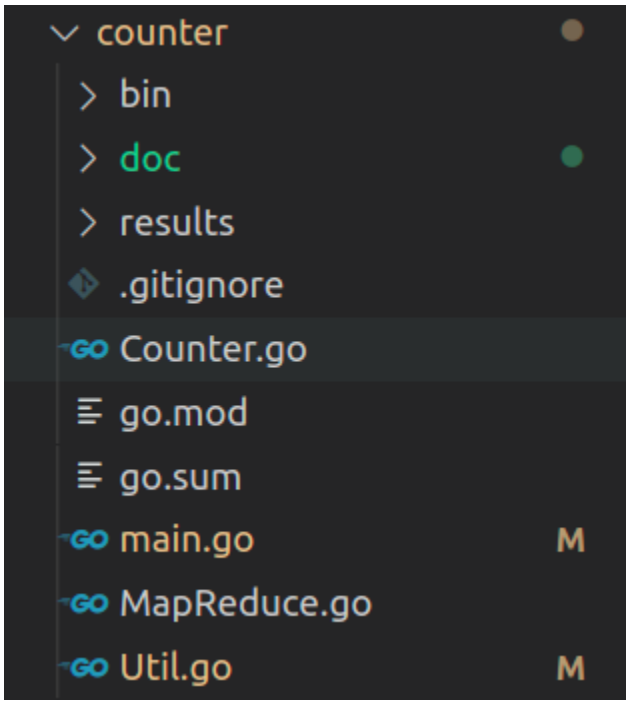
Şekil 5.1 Kelime Sayıcı pseudo kodu

Kelime sayıcı MapReduce algoritması kullanılarak Go dilinde geliştirilmiştir. Kelime sayıcıda her bir process 64’er baytlık bufferlar halinde dosyadan okuma gerçekleştirirler. Her process bir önceki processin okumayı bitirdiği yerin bitişiğinden başlar ve bu sekans böyle ilerler. Her processin ne kadar bayt okuyacağını ise dosyanın boyutu belirlemektedir. Her process dosya boyutu/n kadar bayt okumaktadır.

Okuma işlemi eş zamanlı olarak map işleminde başlatılmaktadır. Tam anlamıyla kelimelerin bölündüğü yerler belirli olmadığı için kelimelerin eksik okunabilmesi problemi söz konusu olmaktadır. Bir kelime tamamen okunabiliyorsa o kelime bir map içerisinde aktarılır. Eksik okunan kelimeler bir önceki veya bir sonraki processlerde eksik okunmuş olan sıradaki kelimeler ile birleştirilerek gerçek kelime elde edilir ve bu da reduce işleminde gerçekleştirilir.

İşlemler tamamlandıktan sonra YAML formatında anlık olarak saate göre bir dosya adı oluşturularak dosyaya kaydedilmektedir.

Kelime sayıcı aracının dosya yapısı Şekil 5.2 üzerinde görülmektedir.



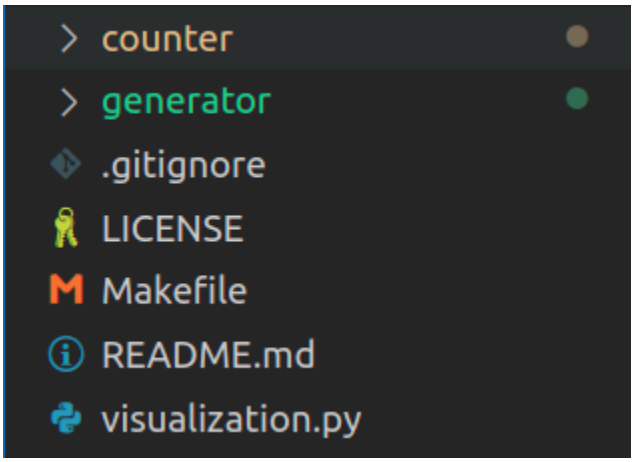
Şekil 5.2 Kelime Sayıcı dosya yapısı

- /bin dizini altında üretilen çalıştırılabilir dosyalar bulunmaktadır.
- /doc dizini altında gerekli dökümanlar bulunmaktadır.
- /results dizini altında üretilmiş olan veri setleri mevcuttur.
- /. dizini altında go paket kodları bulunmaktadır.

6. PROJE YAPISI

Proje iki adet araçtan oluşmaktadır. Bunlar kelime sayıcı ve kelime üreticidir. Kelime üretici belirli parametreler eşliğinde bir dosya üretir kelime sayıcı ise kelime üreticinin ürettiği en son dosya üzerindeki kelimeleri sayarak yeni bir dosya üretir. Tüm işlemler paralel olarak gerçekleştirilmektedir.

Dosya yapısı Şekil 6.1 üzerinde görülmektedir.



Şekil 6.1 Proje dosya yapısı

- /counter dizini altında kelime sayıcı ile ilgili kodlar, dökümanlar vb. dosyalar bulunmaktadır.
- /genarator dizini altında kelime üretici ile ilgili kodlar, dökümanlar vb. dosyalar bulunmaktadır.
- Makefile dosyasında kullanımı kolaylaştıracak kısa betikler bulunmaktadır.
- visualization.py içerisinde 1-100 process aralığında counter veya generator aracına performans benchmarkı yapmanızı sağlayan bir betik bulunmaktadır.

7. GEREKSİNİMLER VE KURULUM

Tavsiye edilen işletim sistemi Ubuntu 20.04'tür. Tüm gereksinimler, kurulum ve çalıştırma işlemleri de bu işletim sistemine göre anlatılmıştır.

Counter:
- Go -> ^1.18.3

Generator:
- C++ -> ^11
- OpenMPI -> ^4.0.3

Make:
- make -> ^4.2.1

Öncelikle güncellemelerinizi kontrol edin ve gerekli güncellemeleri gerçekleştirin.

```
sudo apt-get update && sudo apt-get upgrade
```

C++ 20.04 üzerinde mevcut gelmektedir. Aşağıdaki komut yardımıyla kontrol edebilirsiniz.

```
g++ --version
```

Eğer hata alıyorsanız aşağıdaki komut yardımıyla kurulum gerçekleştirebilir ve tekrar kontrol edebilirsiniz.

```
sudo apt-get install build-essential
```

Aşağıdaki komut yardımıyla MPI kütüphanesini bilgisayarınıza kurabilirsiniz.

```
sudo apt-get install mpich openmpi-bin
```

Kurulum başarıyla tamamlandıktan sonra aşağıdaki komut ile test edebilirsiniz.

```
mpic++ --version
```

Öncelikle `/tmp` dizinine geçin ve daha sonrasında 1.18.* versiyonunu makinenize indirin.

```
cd /tmp && wget
https://go.dev/dl/go1.18.3.linux-amd64.tar.gz
```

İndirme başarıyla gerçekleştikten sonra varsa eski go dosyalarını silin ve yenisini `/usr/local` dizinine kopyalayın

```
sudo rm -rf /usr/local/go && sudo tar -C /usr/local
-xzf go1.18.3.linux-amd64.tar.gz
```

Kopyalama gerçekleştikten sonra dizini `$PATH` içerisine dahil edin. Bunun kalıcı olması için kullandığınız shellin `rc` dosyasına yapıştırın.

- Bash için dosya dizini: $HOME/.bashrc
- Zsh için dosya dizini: $HOME/.zshrc

```
export PATH=$PATH:/usr/local/go/bin
```

Artık aşağıdaki komut ile shell üzerinden go kullanılabilir olduğunu aşağıdaki komut ile test edebilirsiniz.

```
go version
```

Make 20.04 üzerinde varsayılan olarak kurulu gelmektedir. Eğer ki kuruluysa aşağıdaki komut ile kontrol edebilirsiniz.

```
make --version
```

Eğer ki hata alıyorsanız aşağıdaki komut yardımıyla kurulumu gerçekleştirip tekrar kontrol edebilirsiniz.

```
sudo apt-get install make
```

Tüm adımlar hatasız tamamlandıysa kurulumlar gerçekleştirilmiş demektir. Çalıştırma adımına geçebilirsiniz.

8. MAKEFILE

Çalıştırma açısından kolaylık olması için projenin root dizininde makefile oluşturulmuştur. Oluşturulan makefile sayesinde karmaşık komutlar yazılmasına gerek kalmadan ortam değişkenler ve ayraçlar ile her iki araç çalıştırılabilmektedir.

Makefile dosyası size kolay kullanım sağlayan bir CLI komut seti sunar.

Kullanılabilir değişkenler:

- DATASET_SIZE -> min= 1, max=10M, default=50K
- MAX_STR_LEN -> min=2, max=100, default=10
- MIN_STR_LEN -> min=1, max=100, default=2
- GENERATED_FILE_PATH -> Opsiyoneldir, doğru bir dosya yolu olmalı

WORLD_SIZE

Kullanılacak process sayısını belirtir. Hem Generator hem de Counter için geçerlidir.

Kısıtlar: 1 <= X <= 100, varsayılan değer=1

- DATASET_SIZE değişkeninden küçük olamaz.

DATASET_SIZE

Üretilmek istenen kelime sayısını belirtir. Yalnızca Generator tarafından kullanılan bir değişkendir.

Kısıtlar: 1 <= X <= 1000000, varsayılan değer=50000

- WORLD_SIZE değişkeninden büyük olamaz.

MAX_STR_LEN

Üretilecek kelimelerin maksimum sahip olabileceği uzunluğunu belirtir. Yalnızca Generator tarafından kullanılan bir değişkendir.

Kısıtlar: 2 <= X <= 100, varsayılan değer=10

- MIN_STR_LEN değişkeninden küçük olamaz.

MIN_STR_LEN

Üretilecek kelimelerin minimum sahip olabileceği uzunluğunu belirtir. Yalnızca Generator tarafından kullanılan bir değişkendir.

Kısıtlar: 1 <= X <= 100, varsayılan değer=2

- MAX_STR_LEN değişkeninden büyük olamaz.

GENERATED_FILE_PATH

Kelimelerin sayılması istenen dosyanın yolunu özel olarak belirtmek için kullanılabilir. Opsiyoneldir. Default değeri Generator tarafından üretilmiş olan en son dosyadır. Yalnızca Counter tarafından kullanılan bir değişkendir.

Kısıtlar: X > 0, varsayılan değer=son üretilen dosya

9. ÇALIŞTIRMA

Her iki uygulamayı da son hale göre derlemek ve varsayılan değerler ile çalıştırmak isterseniz aşağıdaki komutu deponun root dizininde çalıştırabilirsiniz.

```
make
```

Uygulamaları derledikten sonra özel değerler ile çalıştırmak için

```
make MAX_STR_LEN=15 MIN_STR_LEN=5 WORLD_SIZE=10 DATASET_SIZE=20
```

Uygulamaları özel değerler ile çalıştırmak için(önceden derlenmiş olmalı)

```
make runner MAX_STR_LEN=15 MIN_STR_LEN=5 WORLD_SIZE=10 DATASET_SIZE=20
```

Uygulamaları sadece derlemek için

```
make builder
```

Sadece Generator uygulamasını derlemek için

```
make build_generator
```

Sadece Counter uygulamasını derlemek için

```
make build_counter
```

Sadece Generator uygulamasını çalıştırmak için(önceden derlenmiş olmalı)

```
make run_generator MAX_STR_LEN=15 MIN_STR_LEN=5 WORLD_SIZE=10 DATASET_SIZE=20
```

Sadece Generator uygulamasını derlemek ve çalıştırmak için

```
make BR_generator MAX_STR_LEN=15 MIN_STR_LEN=5 WORLD_SIZE=10 DATASET_SIZE=20
```

Sadece Counter uygulamasını çalıştırmak için(önceden derlenmiş olmalı)

```
make run_counter WORLD_SIZE=10
```

Sadece Counter uygulamasını özel bir dosya yolu vererek çalıştırmak için(önceden derlenmiş olmalı)

```
make run_counter WORLD_SIZE=5 GENERATED_FILE_PATH=$HOME/MPI-words/generator/results/2022_25_05-17_42_11.txt
```

Sadece Counter uygulamasını derlemek ve çalıştırmak için

```
make BR_counter WORLD_SIZE=10
```

10. DEĞERLENDİRME

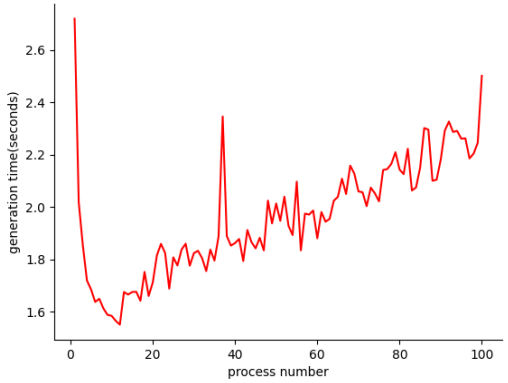
Ölçümlerin yapıldığı cihaz bilgileri:

- OS: Ubuntu 20.04.4 LTS x86_64
- Kernel: 5.13.0-44-generic
- Shell: zsh 5.8
- Terminal: gnome-terminal
- CPU: 11th Gen Intel i5-11400H
- Memory: 16GB DDR4 3200 MHz

NOT: Ölçümler C++ ve Go dillerinin farklılıklarına, çalıştırılabilir dosyalara verdiğimiz parametrelere, ölçümlerin yapıldığı yani bilgisayarımın o anlardaki statelerine göre değişkenlik gösteriyor.

Kelime Oluşturucu

Şekil 10.1 üzerinde 100 uzunluğunda 1000000 adet RASTGELE kelime üreten bir generatorun 1-100 arasında değişen process sayısına bağlı olarak zamana göre değişimini inceleyebilirsiniz.

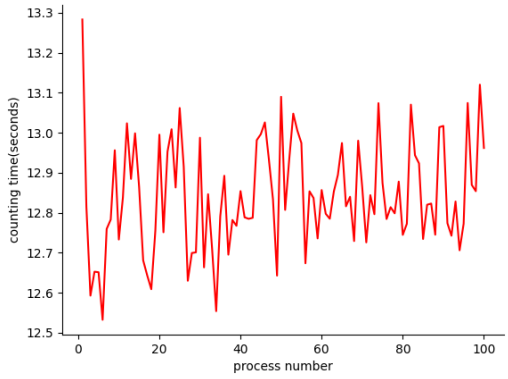


Şekil 10.1 Kelime oluşturucu performansı

- Sonuç olarak 17 process ile paralel çalıştıktan sonra parallelliği arttırmanın bu parametreler düzeyinde yararlı değil aksine zararlı olduğu görülüyor

Kelime Sayıcı

Şekil 10.2 üzerinde 100 uzunluğunda 1000000 adeet kelimeye sahip bir dosyadan kelime sayma işlemini gerçekleştiren ve YAML dosyası olarak kaydeden bir counterın 1-100 aralığında değişen process sayısına bağlı olarak zamana göre değişimini inceleyebilirsiniz.



Şekil 10.2 Kelime sayıcı performansı

- Generator grafiğine göre düzenli bir artış söz konusu değil.
- Bu parametreler eşliğinde bu sefer 7 process ile paralel çalıştırıldıktan sonra daha da paralelleşmenin yararlı değil yine aksine zararlı olduğu görülüyor.

11. REFERANSLAR

- https://www.open-mpi.org/doc/
- https://go.dev/ref/spec
- https://go.dev/doc/
- https://www.youtube.com/channel/UCTE5Fk6-JTvOoN23jwQhH6w/featured
- https://go.dev/tour/concurrency/1
- https://github.com/fatiiates/MPI-words
- https://en.wikipedia.org/wiki/Multiple_instruction,_multiple_data
- https://en.wikipedia.org/wiki/MapReduce

NOT: Derste sunum yapıldı