Turkish Community

Cache Friendly Programming

Oğuzhan KATLI

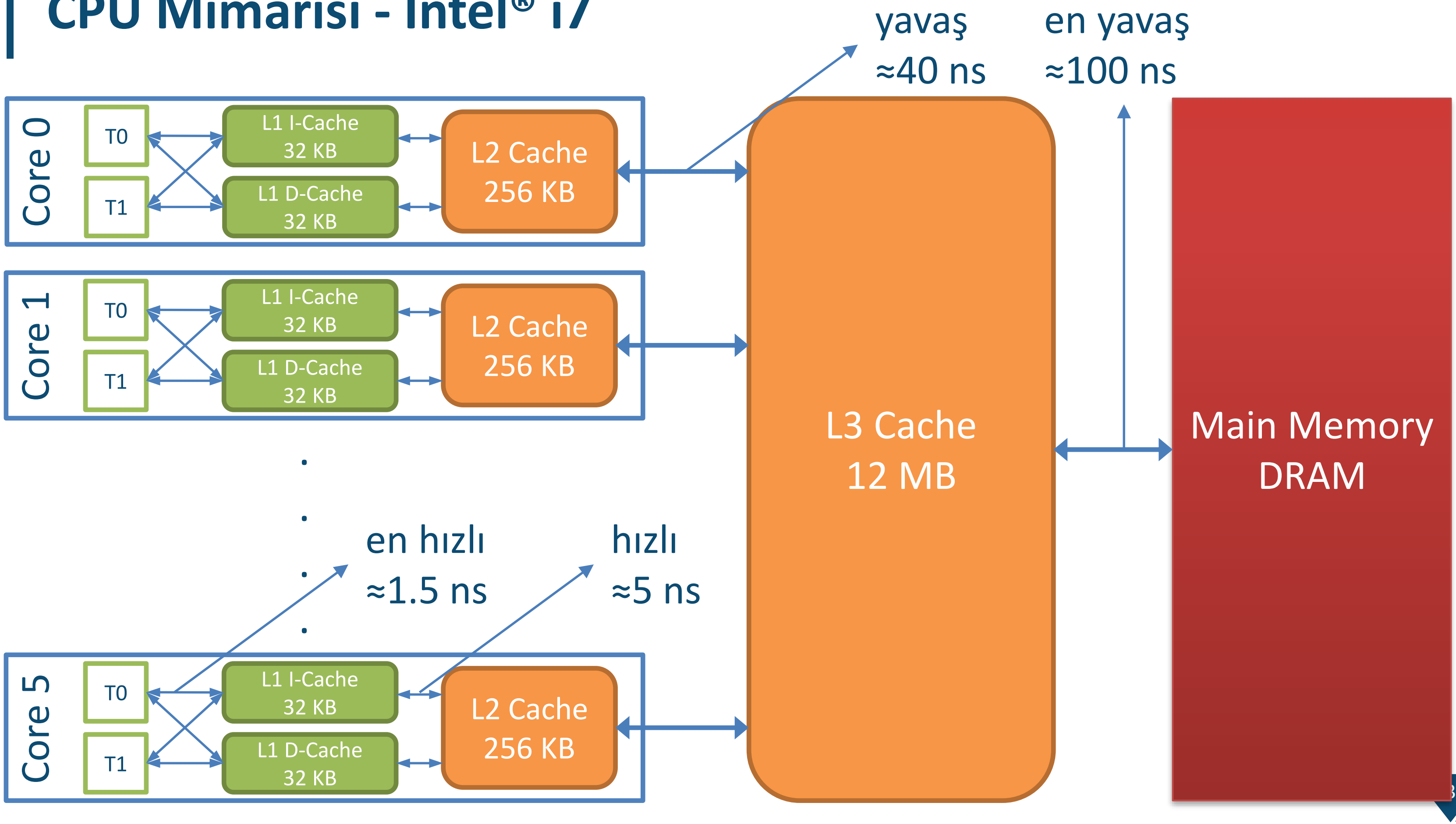
https://www.linkedin.com/in/ogzhnktl

https://github.com/nixiz

Benchmark Setup

- Intel® Core™ i7-9750H İşlemci (12M Önbellek, 4,50 GHz'e kadar)
  - 6 Core – 12 Threads
  - 32 KB L1 Cache
  - 256 KB L2, 12 MB L3 Cache
- Windows 10 Home
  - Visual Studio 2019 Version 16.7.2 – MSVC 19.27
- Ubuntu 20.04 LTS
  - GCC 9.3.0
  - Clang 10.0
- Google Benchmark: https://github.com/google/benchmark
- Benchmark Kodları: https://gist.github.com/nixiz/c3bb8f6029b64f9281ac44cbc119209a

CPU Mimarisi - Intel® i7



Core 0

T0

T1

L1 I-Cache
32 KB

L1 D-Cache
32 KB

L2 Cache
256 KB

Core 1

T0

T1

L1 I-Cache
32 KB

L1 D-Cache
32 KB

L2 Cache
256 KB

Core 5

T0

T1

L1 I-Cache
32 KB

L1 D-Cache
32 KB

L2 Cache
256 KB

L3 Cache
12 MB

Main Memory
DRAM

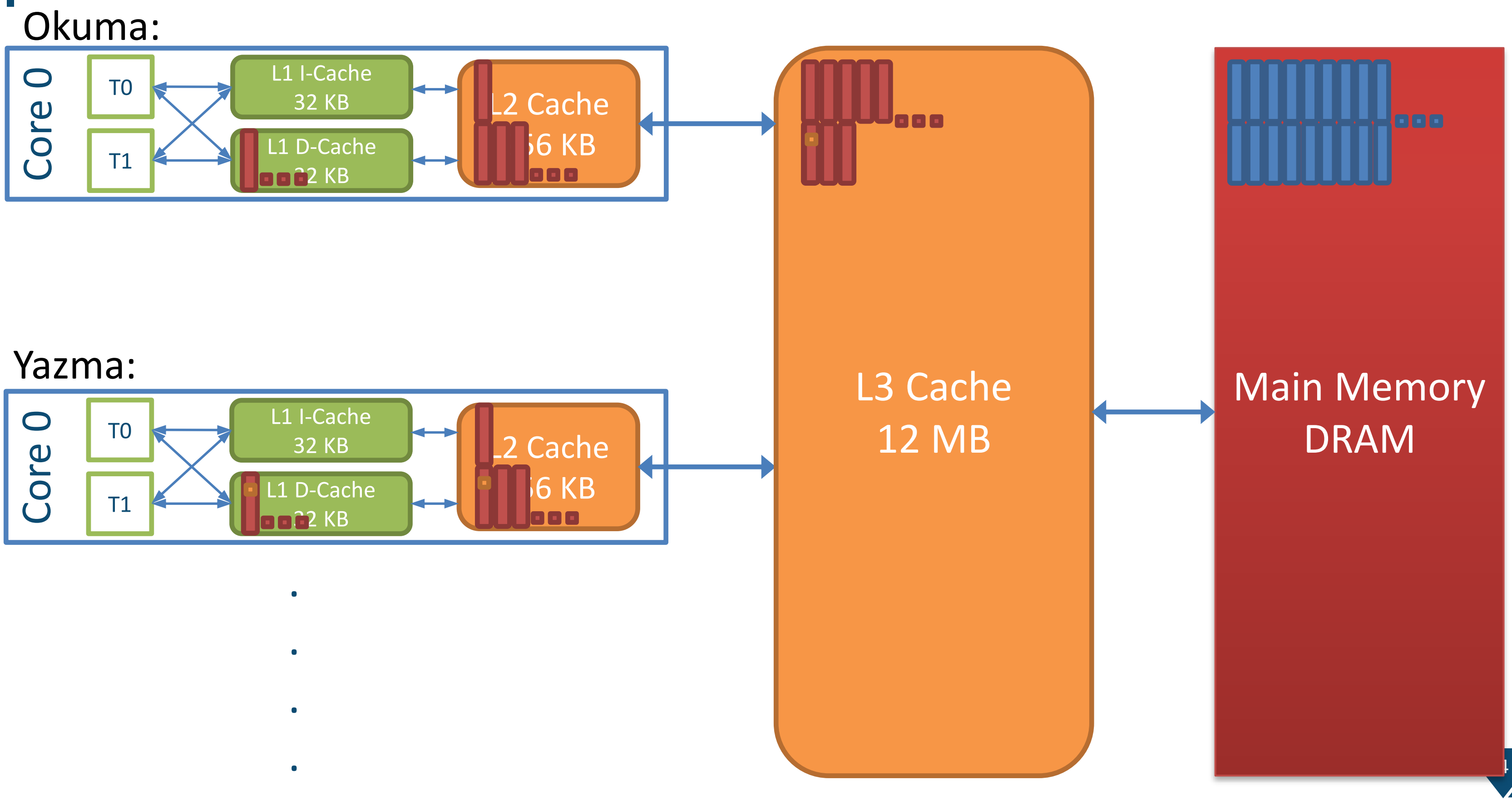
en hızlı
≈1.5 ns

hızlı
≈5 ns

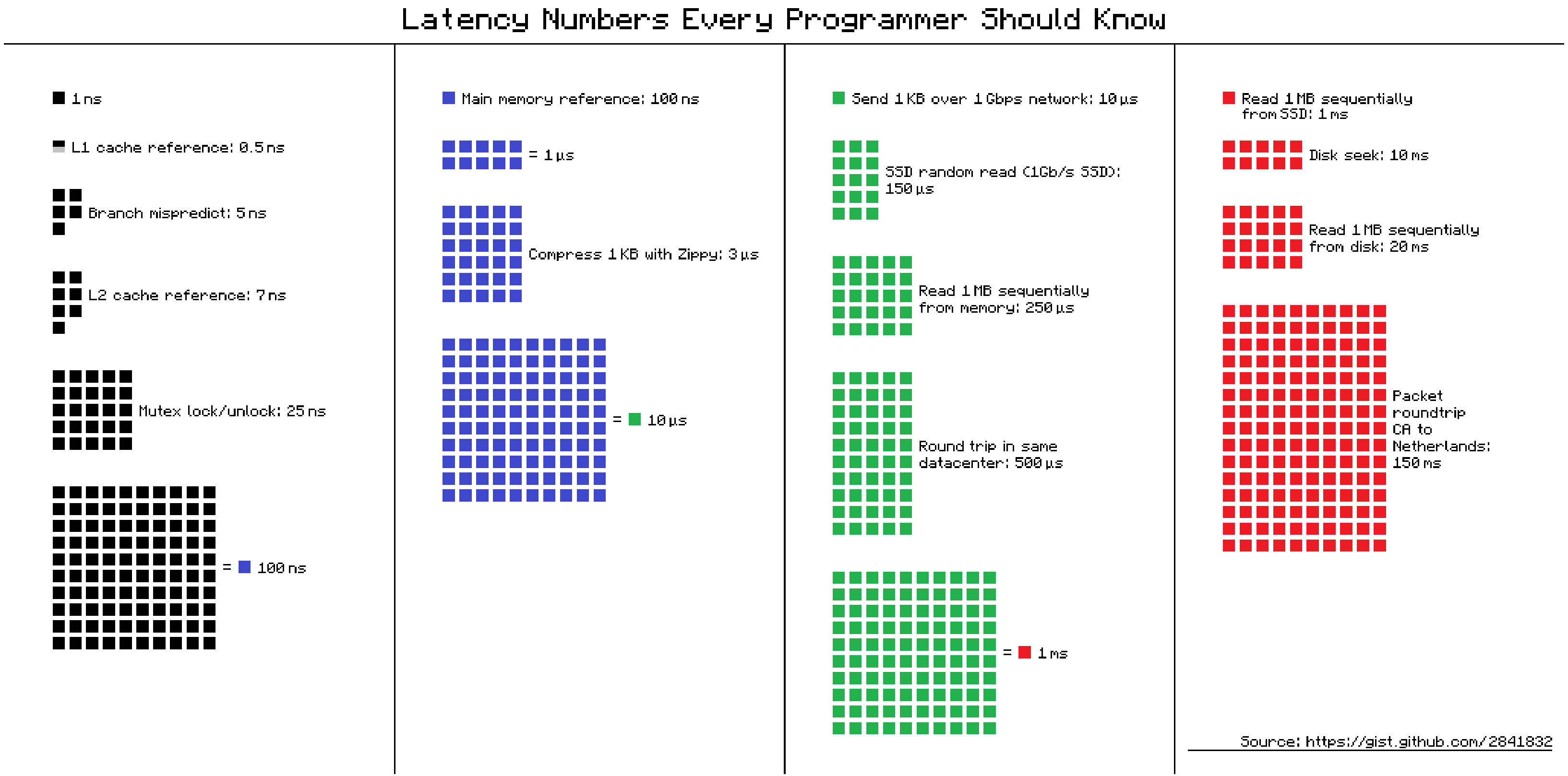
yavaş
≈40 ns

en yavaş
≈100 ns

CPU Mimarisi – Cache Line



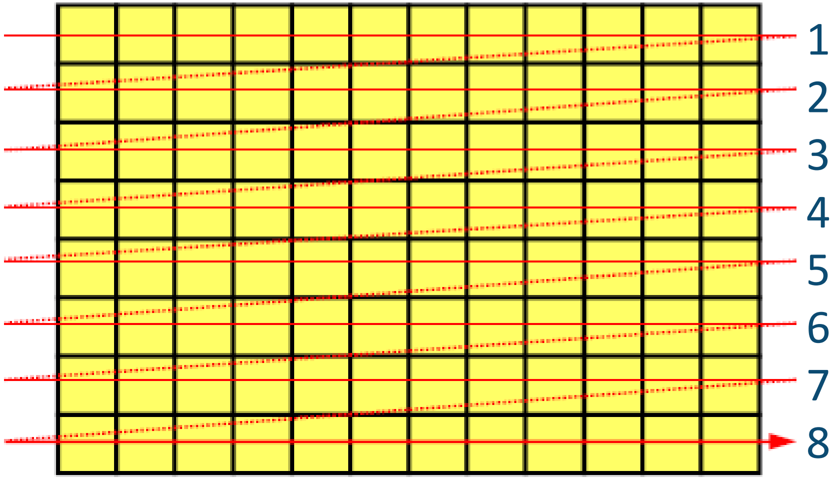
Veri Erişim Hızları



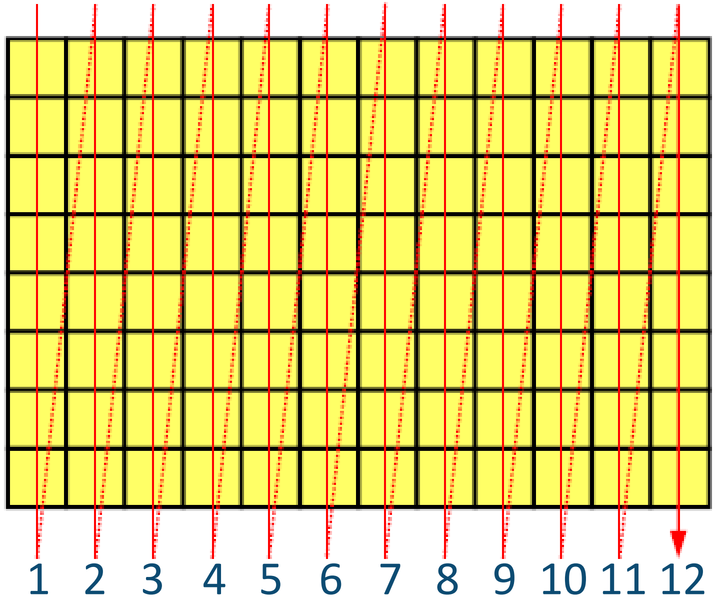
Source: https://gist.github.com/2841832

Row vs Column
Matrix Traversal

Row vs Column Major



Row Major

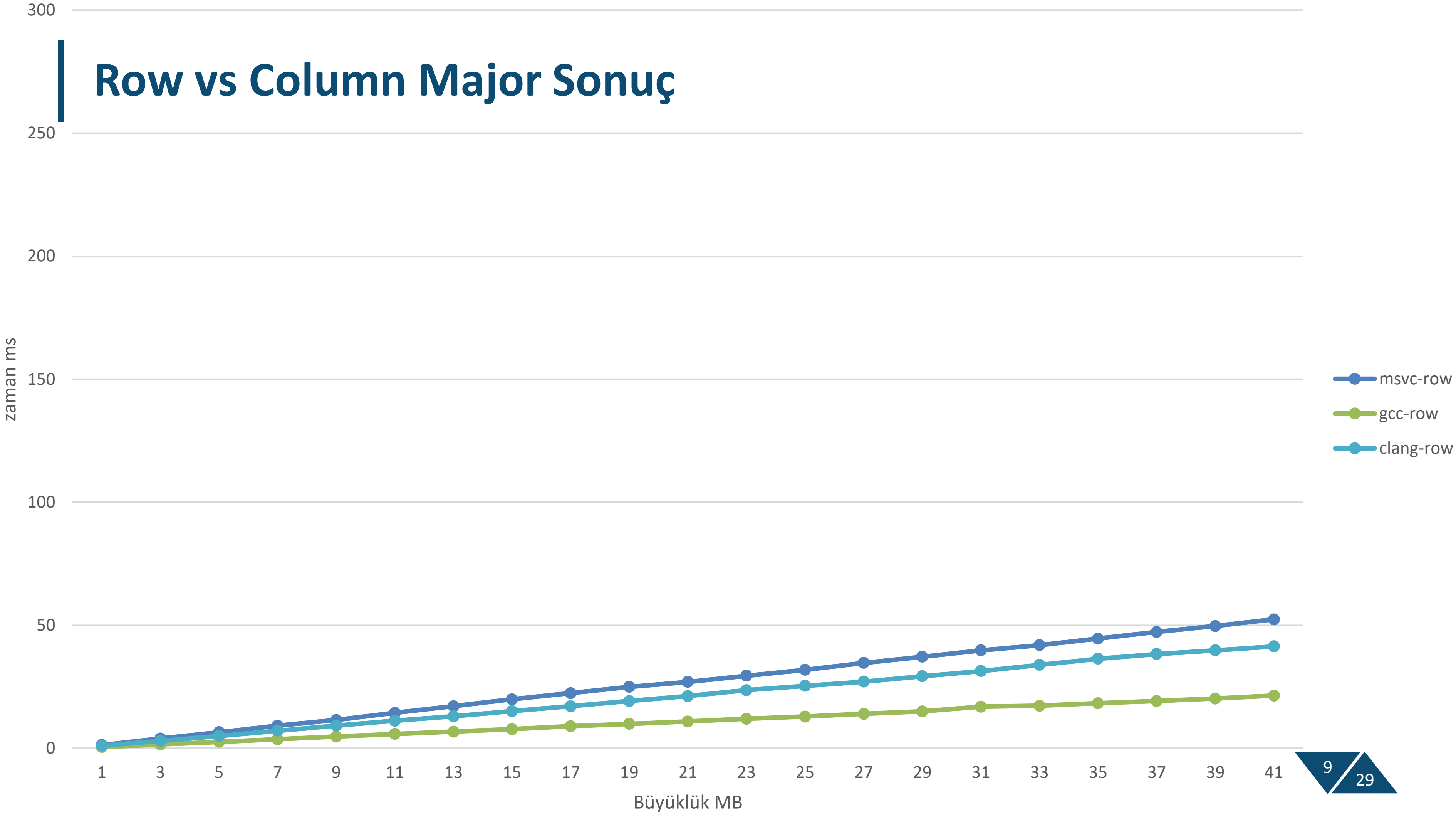


Column Major

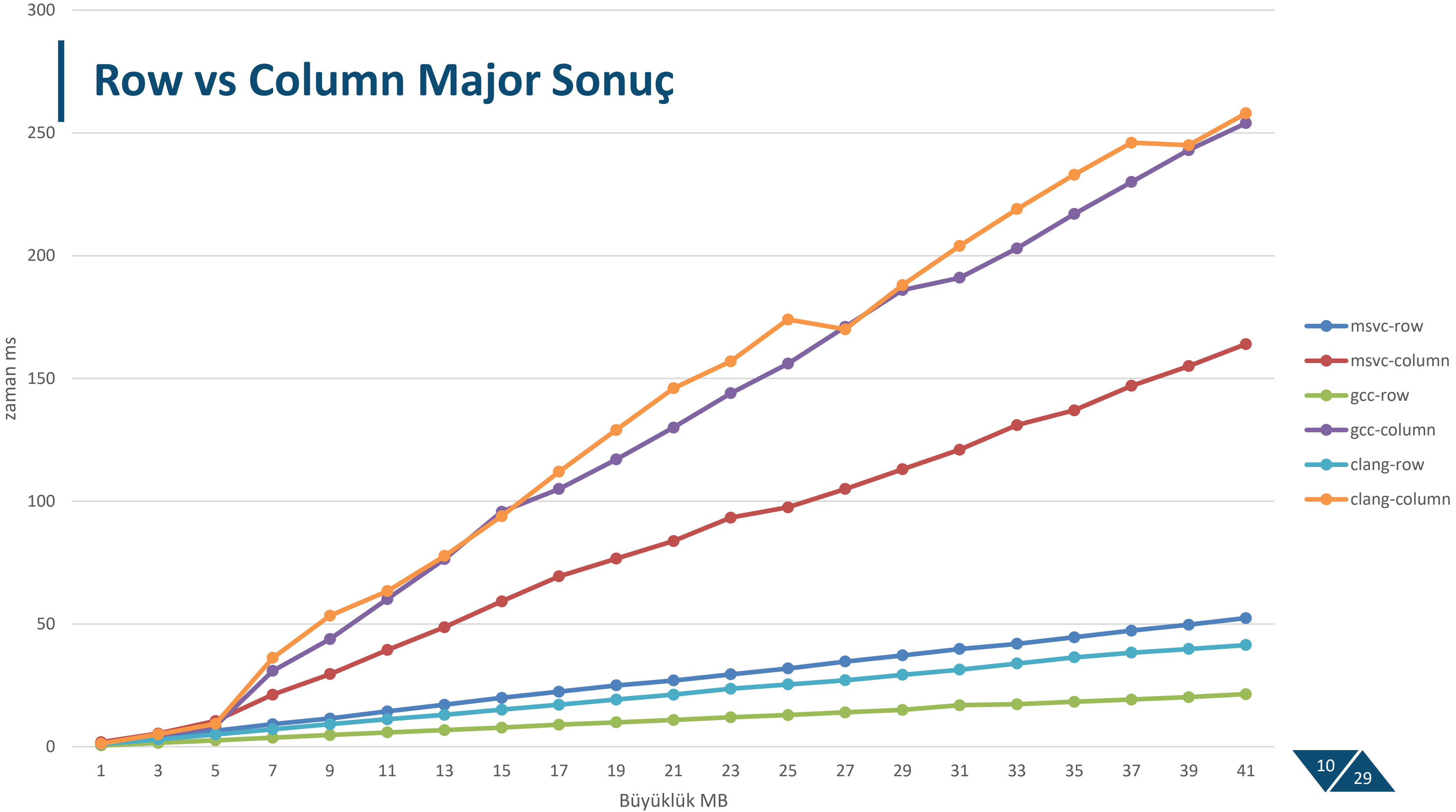
Row vs Column Major

```
template <typename T>
auto sumMatrix(const Matrix<T>& m, TraversalOrder order)
{
  unsigned long sum = 0;
  if (order == TraversalOrder::RowMajor)
  {
    for (auto r = 0; r < m.rows(); ++r)
      for (auto c = 0; c < m.columns(); ++c)
        sum += m[r][c];
  }
  else      // TraversalOrder::ColumnMajor
  {
    for (auto c = 0; c < m.columns(); ++c)
      for (auto r = 0; r < m.rows(); ++r)
        sum += m[r][c];
  }
  return sum;
}
```

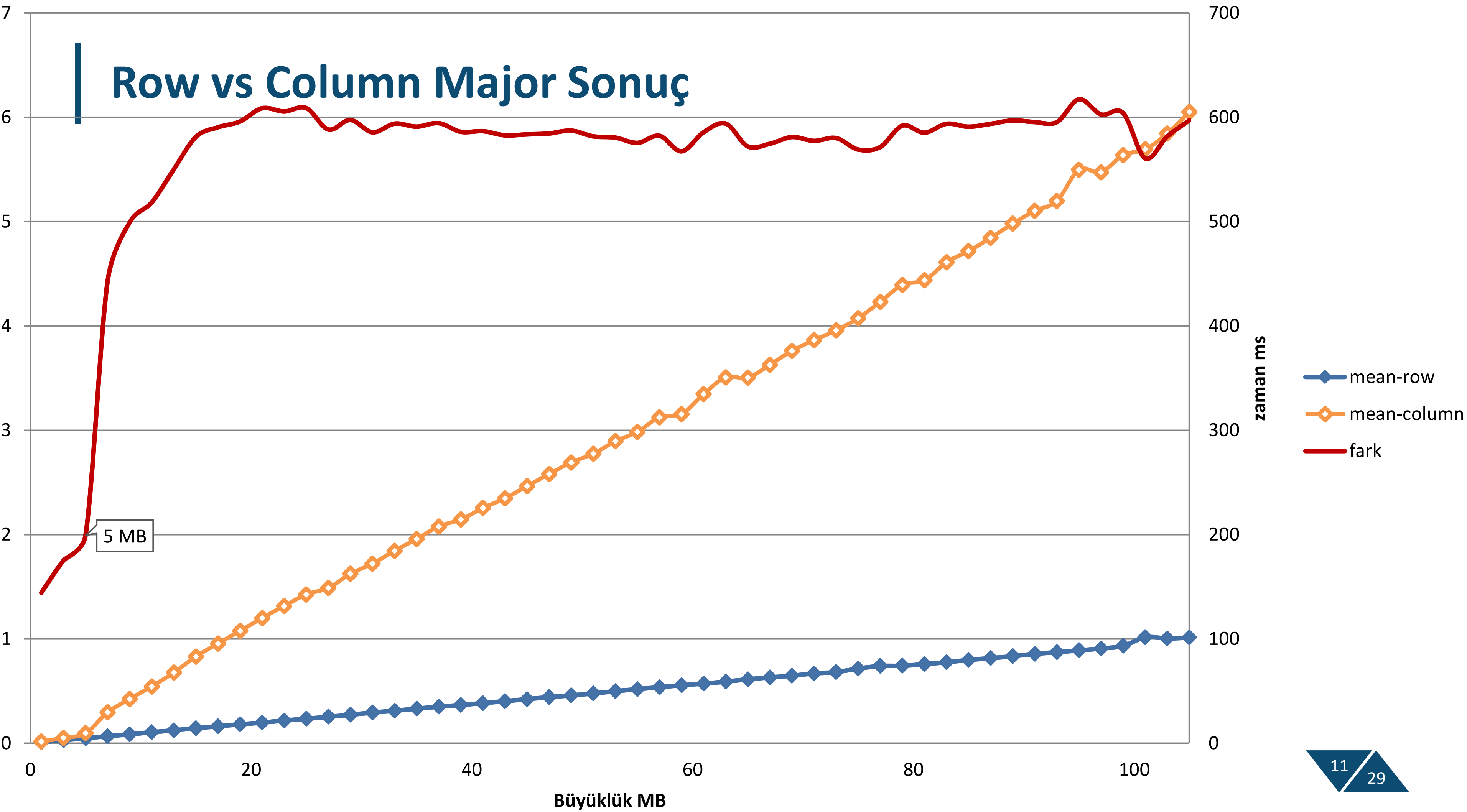
Row vs Column Major Sonuç



Row vs Column Major Sonuç



Row vs Column Major Sonuç



Branch Prediction

Branch Prediction

v:	-1	-1	1	1	-1	1	-1	-1	1	-1	-1	1	1	-1	1
v:	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1

```
std::vector<float> v(65536);
std::generate(std::begin(v), std::end(v), [] {
  return (rand() % 2) ? 1 : -1;
});

std::sort(v.begin(), v.end());

return std::count_if(std::begin(v), std::end(v), [](float x) {
  return x > 0;
});
```

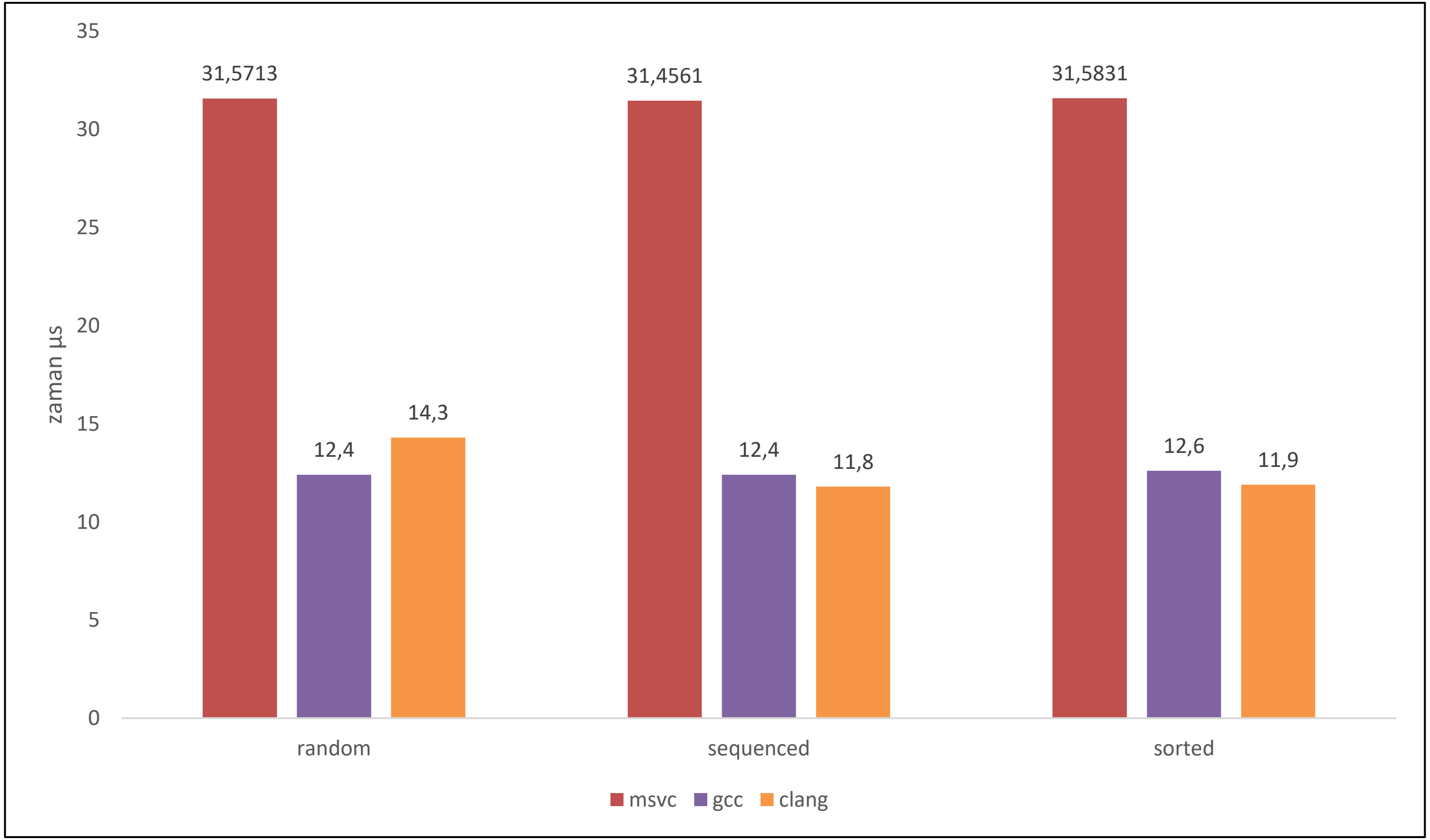
Branch Prediction – Sequenced If - Else

v:	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1

```
std::vector<float> v(65536);
std::generate(std::begin(v), std::end(v), [n = 0]() mutable {
    return (++n % 2) ? 1 : -1;
  });


return std::count_if(std::begin(v), std::end(v), [](float x) {
  return x > 0;
});
```

Count If Sonuç



Virtual Function Calls

```
struct base_price
{
  virtual ~price() {}
  virtual float getPrice() const noexcept { return 0.0; }
};

struct cheap : public base_price
{
  float getPrice() const noexcept override { return 2.0; }
};

struct expensive : public base_price
{
  float getPrice() const noexcept override { return 3.14159; }
};
```

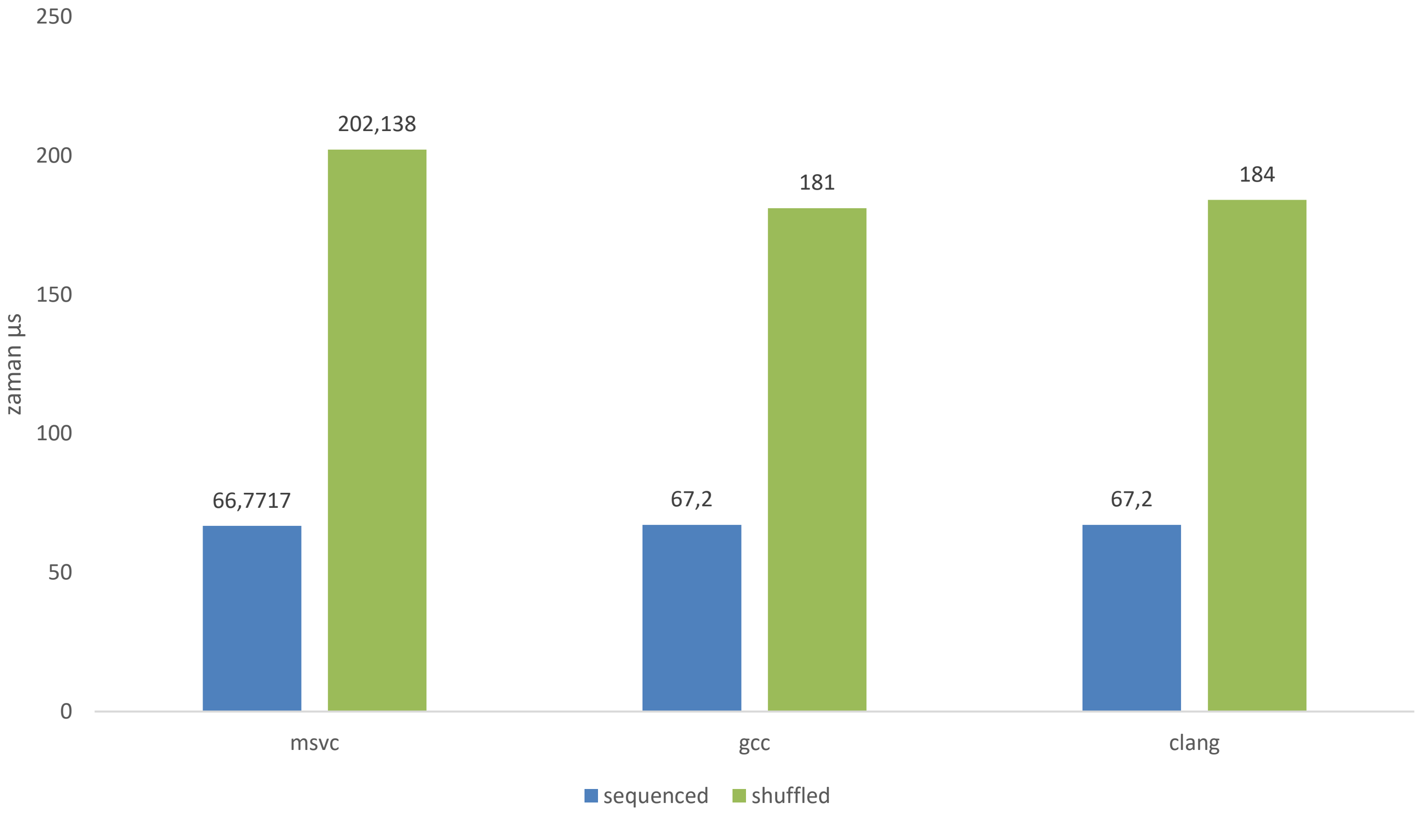
Örnek

```
std::vector<base_price*> pricelist;
std::fill_n(std::back_inserter(pricelist), 10000, new base_price);
std::fill_n(std::back_inserter(pricelist), 10000, new cheap);
std::fill_n(std::back_inserter(pricelist), 10000, new expensive);

std::random_shuffle(pricelist.begin(), pricelist.end());

float sum = 0;
for (auto *p : pricelist)
{
  sum += p->getPrice();
}
```

Virtual Call Sonuç



Cache Friendly Programming

False Sharing

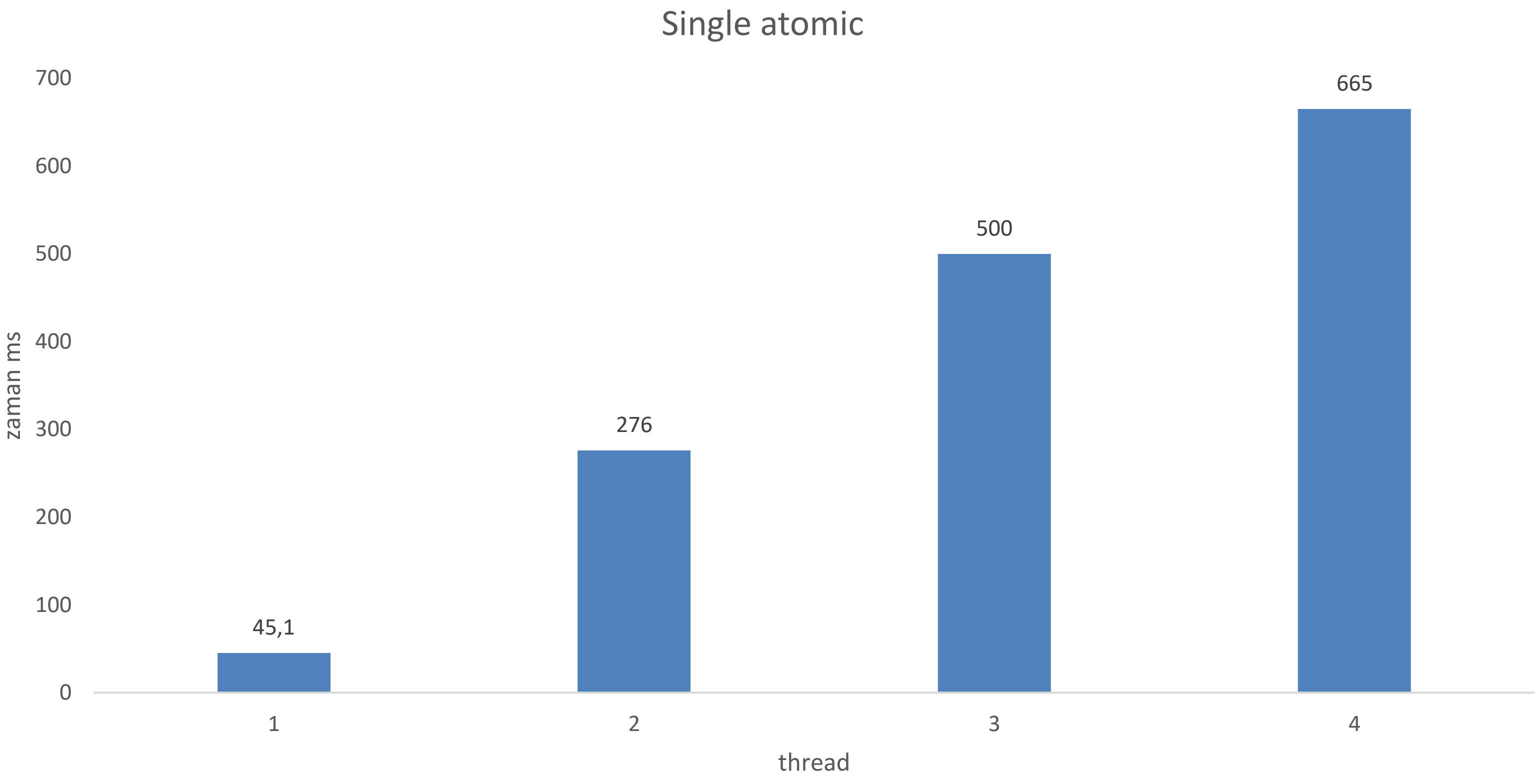
False Sharing

```
template <typename input_t>
void work(input_t& a)
{
  for (int i = 0; i < 10,000,000; ++i)
    a++;
}
int test_with_4_threads()
{
  std::atomic<int> a; a = 0;

  std::thread t1([&] { work(a); });
  std::thread t2([&] { work(a); });
  std::thread t3([&] { work(a); });
  std::thread t4([&] { work(a); });

  t1.join(); t2.join(); t3.join(); t4.join();
  return a;
}
```

Sonuç


Single atomic
700
600
500
400
300
200
100
0
zaman ms
45,1
276
500
665
1
2
3
4
thread

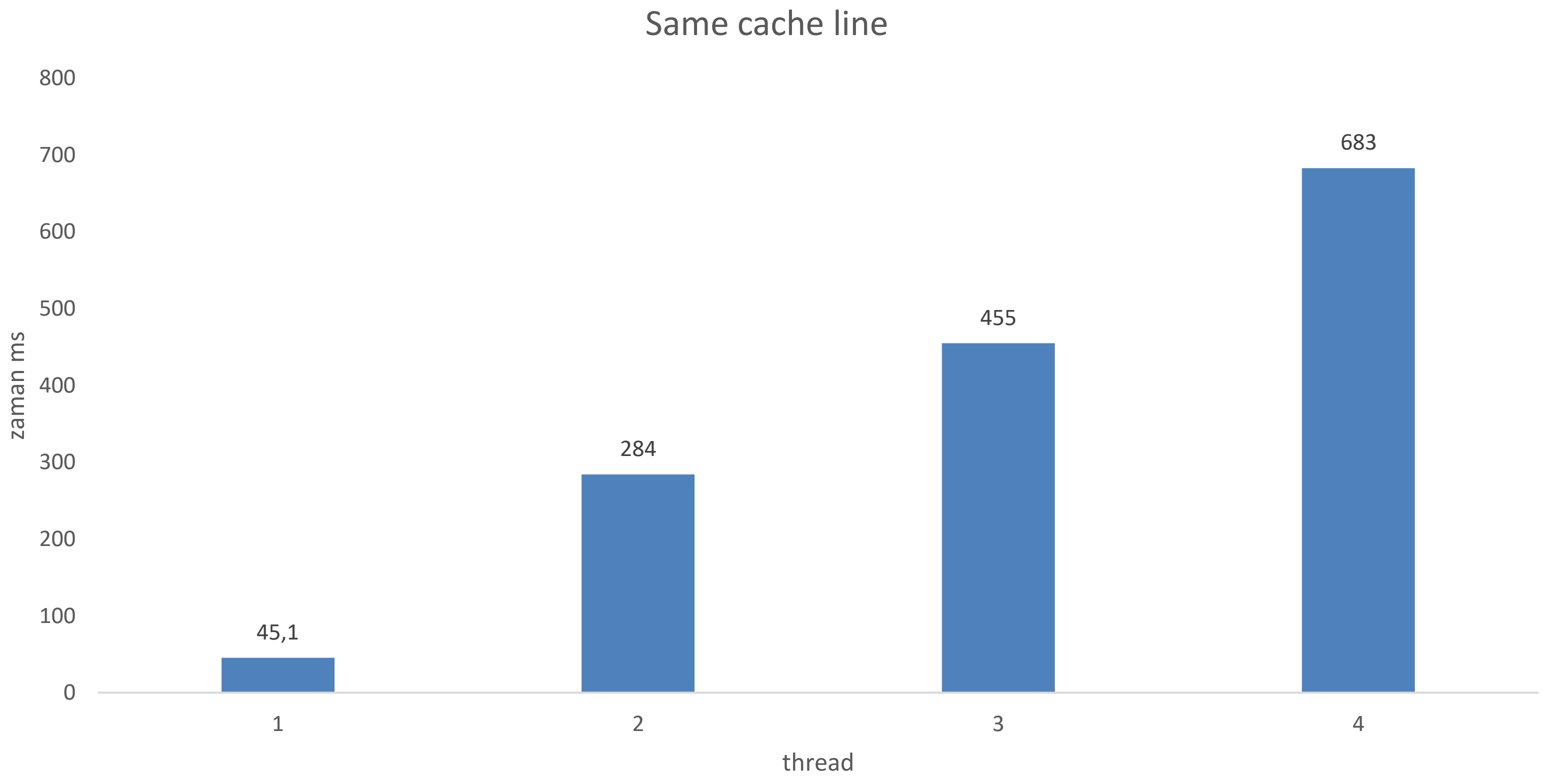
Farklı Atomik Nesneler

```
int test()
{
  std::atomic<int> a; a = 0;
  std::atomic<int> b; b = 0;
  std::atomic<int> c; c = 0;
  std::atomic<int> d; d = 0;

  std::thread t1([&] { work(a); });
  std::thread t2([&] { work(b); });
  std::thread t3([&] { work(c); });
  std::thread t4([&] { work(d); });

  t1.join(); t2.join(); t3.join(); t4.join();
  return a + b + c + d;
}
```

Sonuç


Same cache line
800
700
600
500
400
300
200
100
0
zaman ms
45,1
284
455
683
1
2
3
4
thread

Farklı Atomik Nesneler

Adres blokları aynı

```
int test()
{
  std::atomic<int> a; a = 0;  // &a: 0x...b2f7c0
  std::atomic<int> b; b = 0;  // &b: 0x...b2f7c4
  std::atomic<int> c; c = 0;  // &c: 0x...b2f7c8
  std::atomic<int> d; d = 0;  // &d: 0x...b2f7cc

  std::thread t1([&] { work(a); });
  std::thread t2([&] { work(b); });
  std::thread t3([&] { work(c); });
  std::thread t4([&] { work(d); });

  t1.join(); t2.join(); t3.join(); t4.join();
  return a + b + c + d;
}
```

Çözüm

```
struct alignas(64) aligned_type
{
  std::atomic<int> val;
};
```

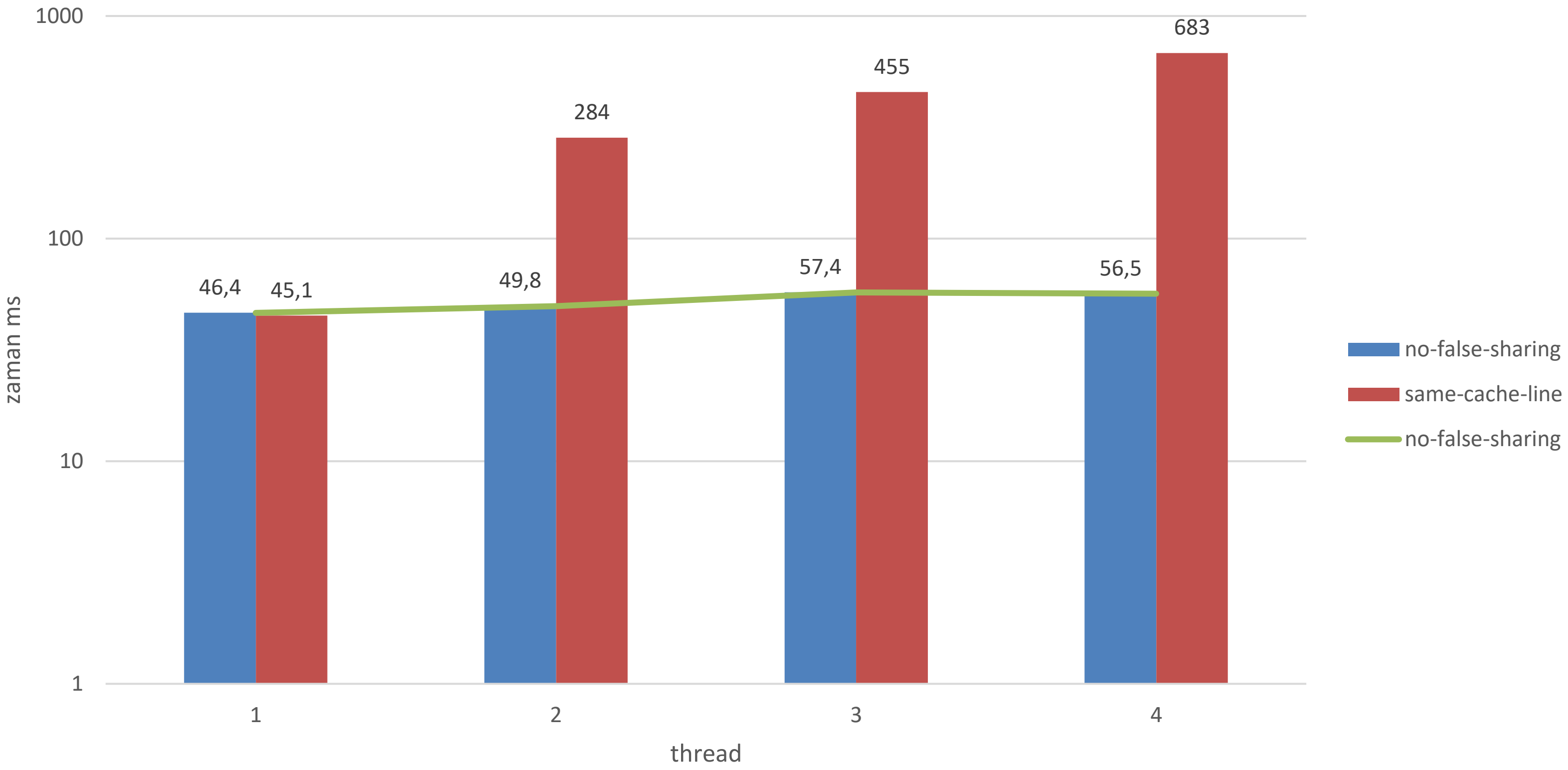
Adres blokları **Farklı**

```
int test()
{
  aligned_type a; a.val = 0; // &a: 0x...4ff240
  aligned_type b; b.val = 0; // &b: 0x...4ff280
  aligned_type c; c.val = 0; // &c: 0x...4ff2c0
  aligned_type d; d.val = 0; // &d: 0x...4ff300

  std::thread t1([&a] { work(a.val); });
  std::thread t2([&b] { work(b.val); });
  std::thread t3([&c] { work(c.val); });
  std::thread t4([&d] { work(d.val); });

  t1.join(); t2.join(); t3.join(); t4.join();
  return a.val + b.val + c.val + d.val;
}
```

Sonuç



1000
100
10
1
zaman ms
46,4
45,1
49,8
284
57,4
455
56,5
683
1
2
3
4
thread
no-false-sharing
same-cache-line
no-false-sharing

False Sharing – Sentinels

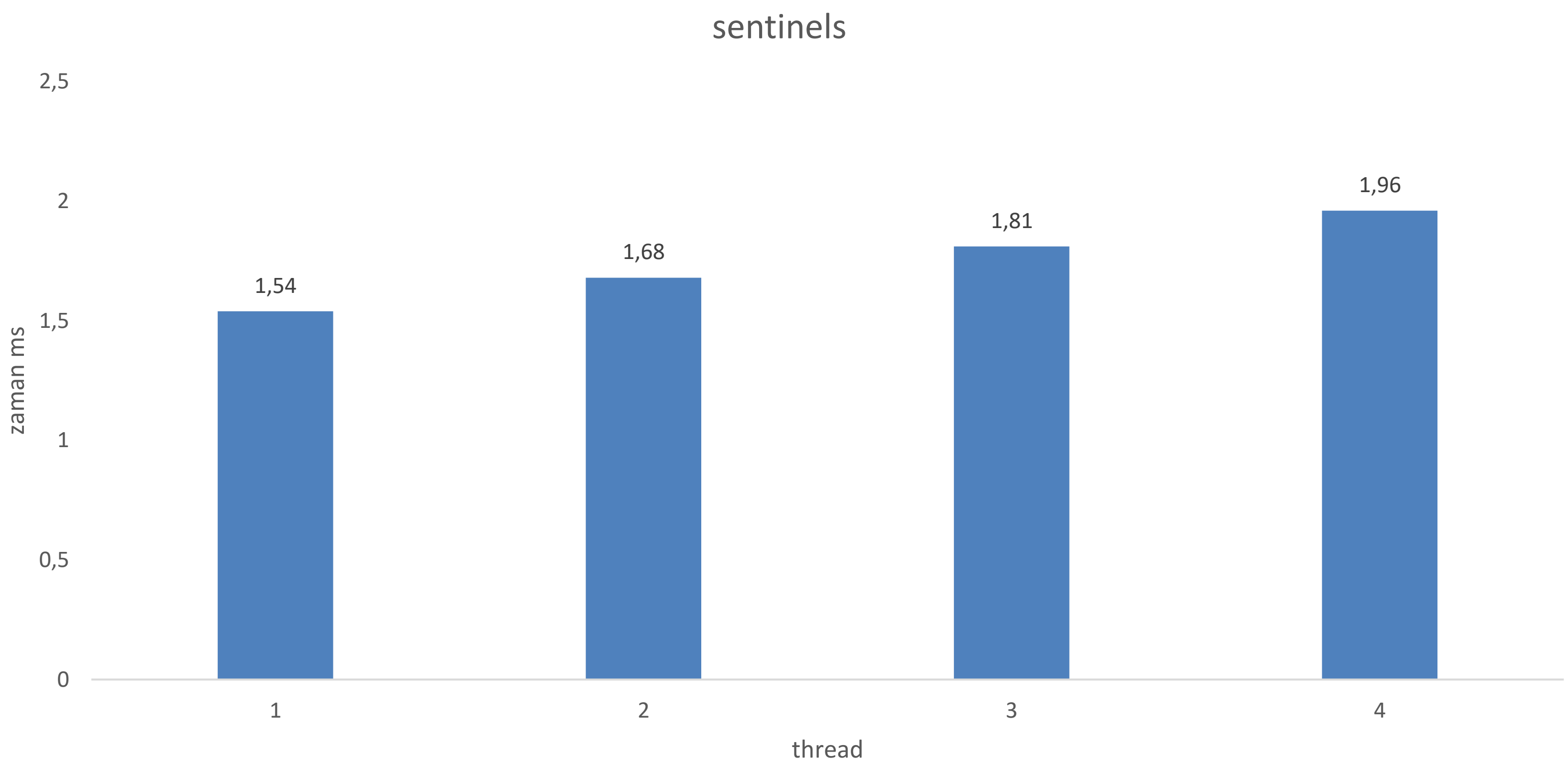
```
void work_with_sentinel(std::atomic<int>& a) {
  thread_local unsigned int sentinel = 0;
  for (int i = 0; i < 10,000,000; ++i)
    ++sentinel;
  a += sentinel;
}

int test_with_sentinels() {
  std::atomic<int> a; a = 0;

  std::thread t1([&] { work_with_sentinel(a); });
  std::thread t2([&] { work_with_sentinel(a); });
  std::thread t3([&] { work_with_sentinel(a); });
  std::thread t4([&] { work_with_sentinel(a); });

  t1.join(); t2.join(); t3.join(); t4.join();
  return a;
}
```

Sonuç



Cache Friendly Programming

Teşekkürler