

C# 00 Programmeren

MD5 COLLISIONS PROFILING

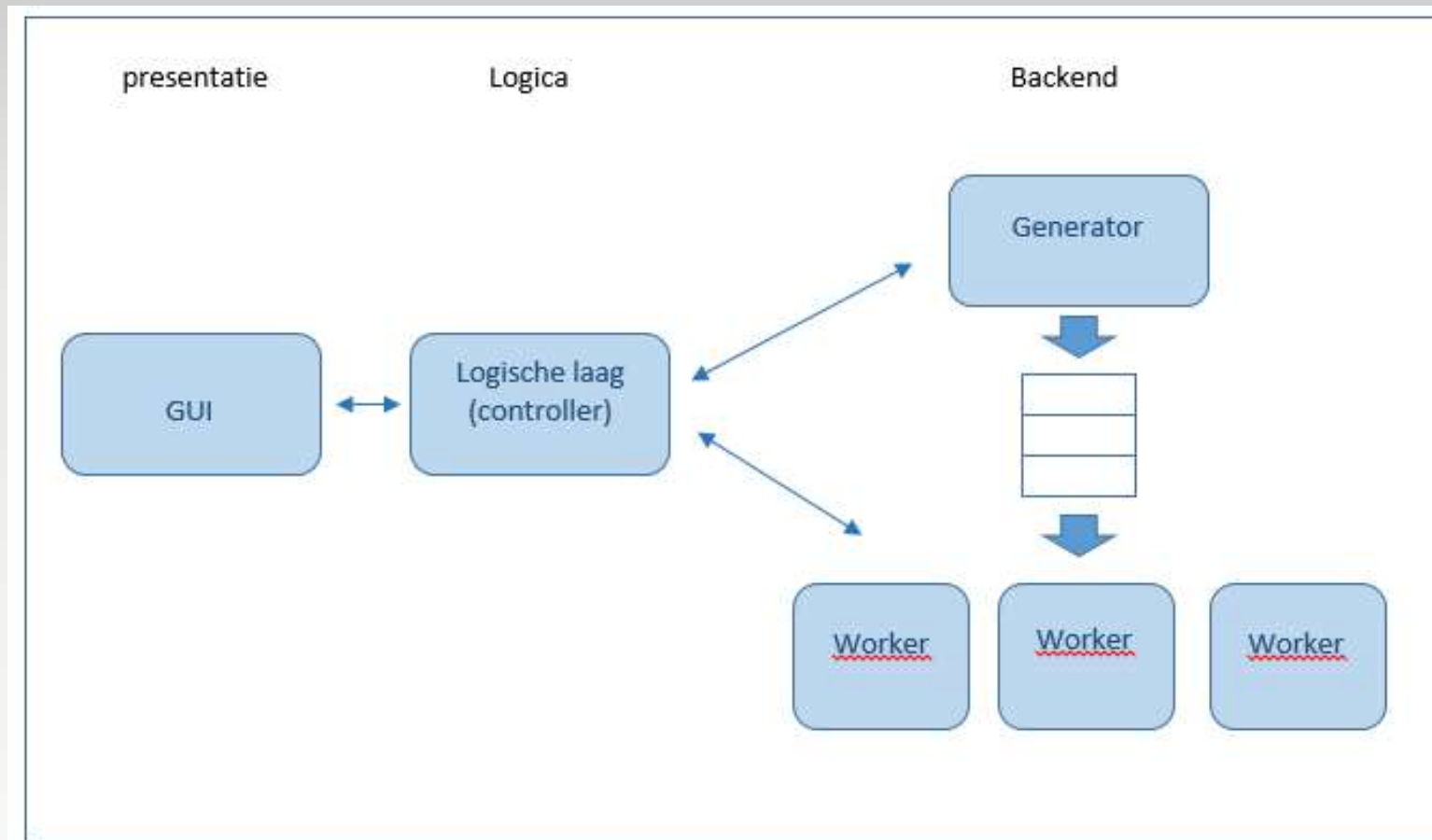
JOHAN DONNÉ

MD5 Collision calculator

The screenshot shows a Windows application window titled "MD5 Collision Calculator". The interface includes several input fields and buttons. The "Password" field contains "ZEVEN", and the "Hash" field contains "783ff203103e4841f771dfc71ca43ab1". A "Collision" field is empty. To the right of these fields are buttons for "Calculate Hash", "Search Collision", and "Abort". Below the "Collision" field is a progress bar showing 21.76% completion. At the bottom, there are several status fields: "Time running" (00:00:00:20), "Time left (max)" (00:00:01:12), "generator stalls/sec" (82), "hashes/sec" (129 011), "worker stalls/sec" (0), and "# combinations" (11 881 376). The "# Workers" field is set to 1.

Field	Value
Password	ZEVEN
Hash	783ff203103e4841f771dfc71ca43ab1
Collision	
# Workers	1
Progress	21.76%
Time running	00:00:00:20
Time left (max)	00:00:01:12
generator stalls/sec	82
hashes/sec	129 011
worker stalls/sec	0
# combinations	11 881 376

architectuur



Backend: Generator



Bedoeling:

- bij 'Start()', maakt de generator een ConcurrentQueue aan
- ... en start een Task die paswoorden genereert en in de queue plaatst
- communicatie gebeurt vanaf dan via methodes en events

Generator

2 delen:

- GUI –thread: alles dat van buiten uit gebruikt wordt + opstarten van background task
- Background task

Aandachtspunten:

- Afbreken background task moet mogelijk zijn (meerdere technieken mogelijk).

Generator

```
public ConcurrentQueue<string> Start(int passWordLength, int maxQueueLength)
{
    pwGenerator = new PasswordGenerator(passWordLength);
    var queue = new ConcurrentQueue<string>();

    cancelSource = new CancellationTokenSource();
    var cancelToken = cancelSource.Token;
    Task.Run(() =>
    {
        GeneratePassWords(queue, pwGenerator, maxQueueLength, cancelToken);
    });
    return queue;
}
```

Generator: timing voor events

Belangrijk: background task moet GUI thread ontlasten.

Dus: GUI – thread niet ‘flooden’ met events.

Dus: enkel met redelijk interval voortgang rapporteren vanuit background task.

```
var time = DateTime.Now;

foreach (var pw in generator)
{
    var thisTime = DateTime.Now;
    if ((thisTime - time).TotalMilliseconds > 50)
    {
        time = thisTime;
        PasswordsGeneratedReport?.Invoke(count);
    }
}
```

Worker

Ook 2 delen:

- GUI –thread: alles dat van buiten uit gebruikt wordt + opstarten van background task
- Background task

Aandachtspunten:

- zelfde als Generator
- bovendien: worker weet niet wanneer werk gedaan is: kan enkel stoppen vanwege een Abort().

Logica: MD5CollisionCalculator

- Generator (her)starten
- Workers aanmaken/verwijderen
- Events van background tasks kanaliseren naar UI
- Abort, Close vanuit UI doorgeven naar background tasks

Generator: 1 x aangemaakt in constructor, (her)starten via Start().

Workers: dynamisch aanmaken, verwijderen (wegwerp objecten)

```
public MD5CollisionCalculator()
{
    nrOfWorkersTasks = 1;
    workerList = new List<IWorker>();
    generator = new Generator();
    generator.GeneratorFinished += OnGeneratorGeneratorFinished;
    generator.Stalled += OnGeneratorStalled;
}
```

MD5CollisionCalculator – berekening starten

```
public void StartCalculatingMD5Collision(string hash, int passwordLength)
{
    this.hash = hash;

    // start generator
    queue = generator.Start(passwordLength, maxQueueLength);
    running = true;
    for (int i = 0; i < NrofWorkerTasks; i++)
    {
        AddWorker();
    }
}

private void AddWorker()
{
    var worker = GetWorker();
    worker.Stalled += OnWorkerStalled;
    worker.CollisionFound += OnWorkerCollisionFound;
    worker.ProgressChanged += OnWorkerProgressChanged;
    worker.GetCollisions(hash, queue);
    workerList.Add(worker);
}
```

MD5CollisionCalculator – afbreken

```
public void Abort()
{
    generator.Abort();
    while (workerList.Count > 0) RemoveWorker();
    timer.Change(Timeout.Infinite, Timeout.Infinite);
    running = false;
}

public void Close()
{
    generator.GeneratorFinished -= OnGeneratorGeneratorFinished;
    generator.Stalled -= OnGeneratorStalled;
    generator.Close();
    foreach (var worker in workerList)
    {
        worker.Stalled -= OnWorkerStalled;
        worker.CollisionFound -= OnWorkerCollisionFound;
        worker.ProgressChanged -= OnWorkerProgressChanged;
        worker.Abort();
    }
}
```

Bij Close: eventhandlers afschakelen !!!

MD5CollisionCalculator – # workers wijzigen

```
public int NrOfWorkersTasks
{
    get
    {
        return nrOfWorkersTasks;
    }

    set
    {
        if (running)
        {
            while (value > nrOfWorkersTasks)
            {
                AddWorker();
                nrOfWorkersTasks++;
            }
            while (value < nrOfWorkersTasks)
            {
                RemoveWorker();
                nrOfWorkersTasks--;
            }
        }
        nrOfWorkersTasks = value;
    }
}
```

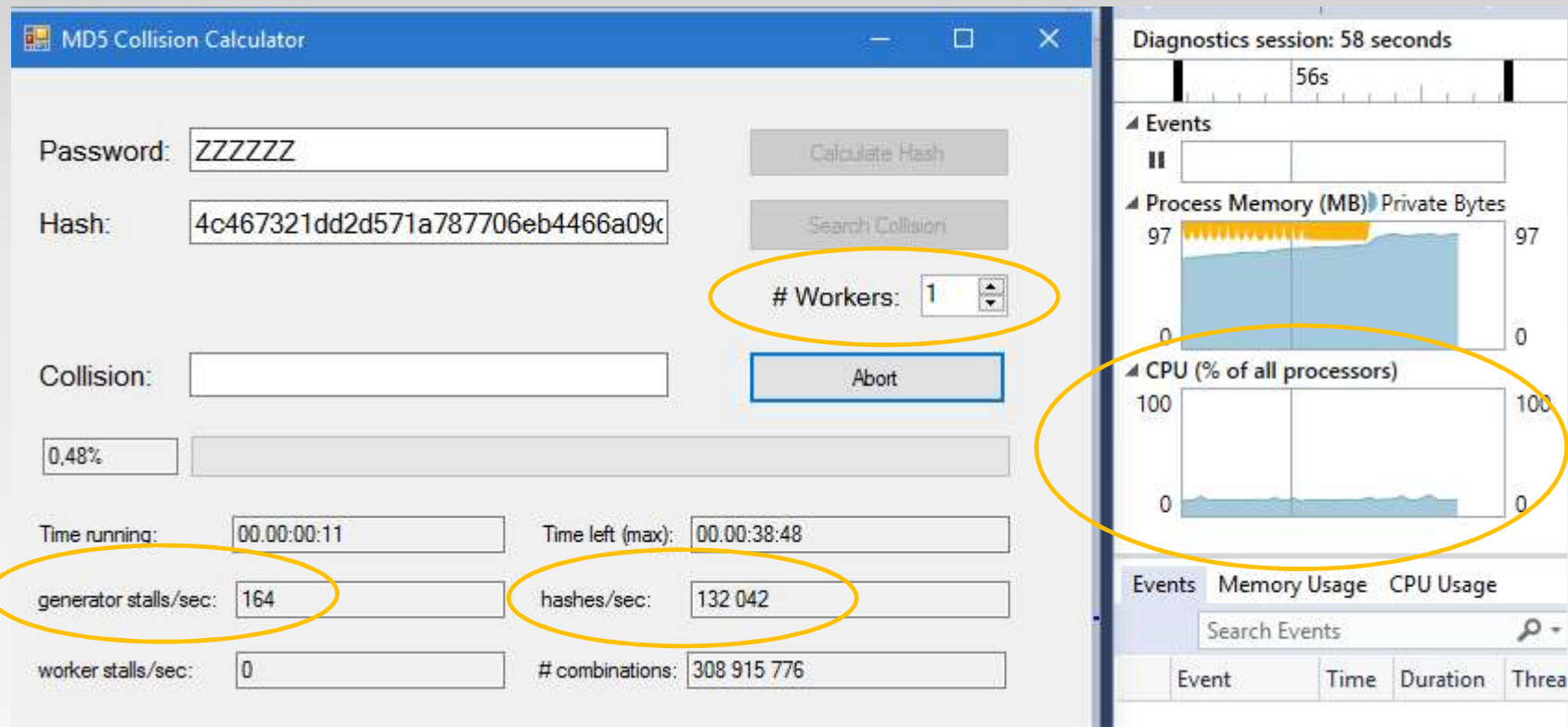
MD5CollisionCalculator – varia

```
private void OnWorkerProgressChanged(ulong hashesCalculated)
{
    lock (mutex)
    {
        nrOfPassWordsProcessed += hashesCalculated;
        ProgressChanged?.Invoke((100M*nrOfPassWordsProcessed)/generator.MaxCount());
    }
}
```

```
private void OnGeneratorGeneratorFinished()
{
    while (queue.Count > 0)
    {
        Thread.Sleep(5);
    }
    Abort();
}
```

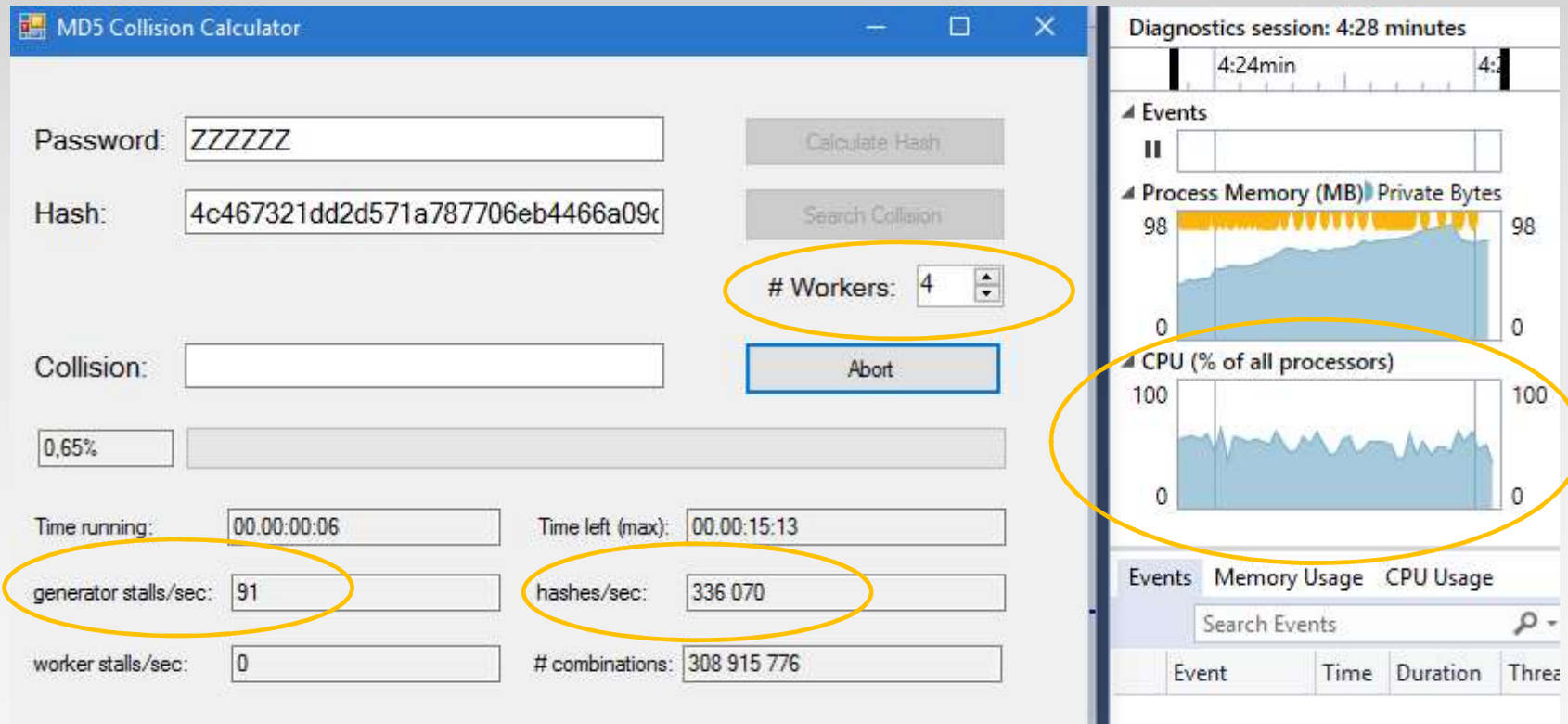
Tenslotte: profiling (geen stof)

1 worker



Tenslotte: profiling (geen stof)

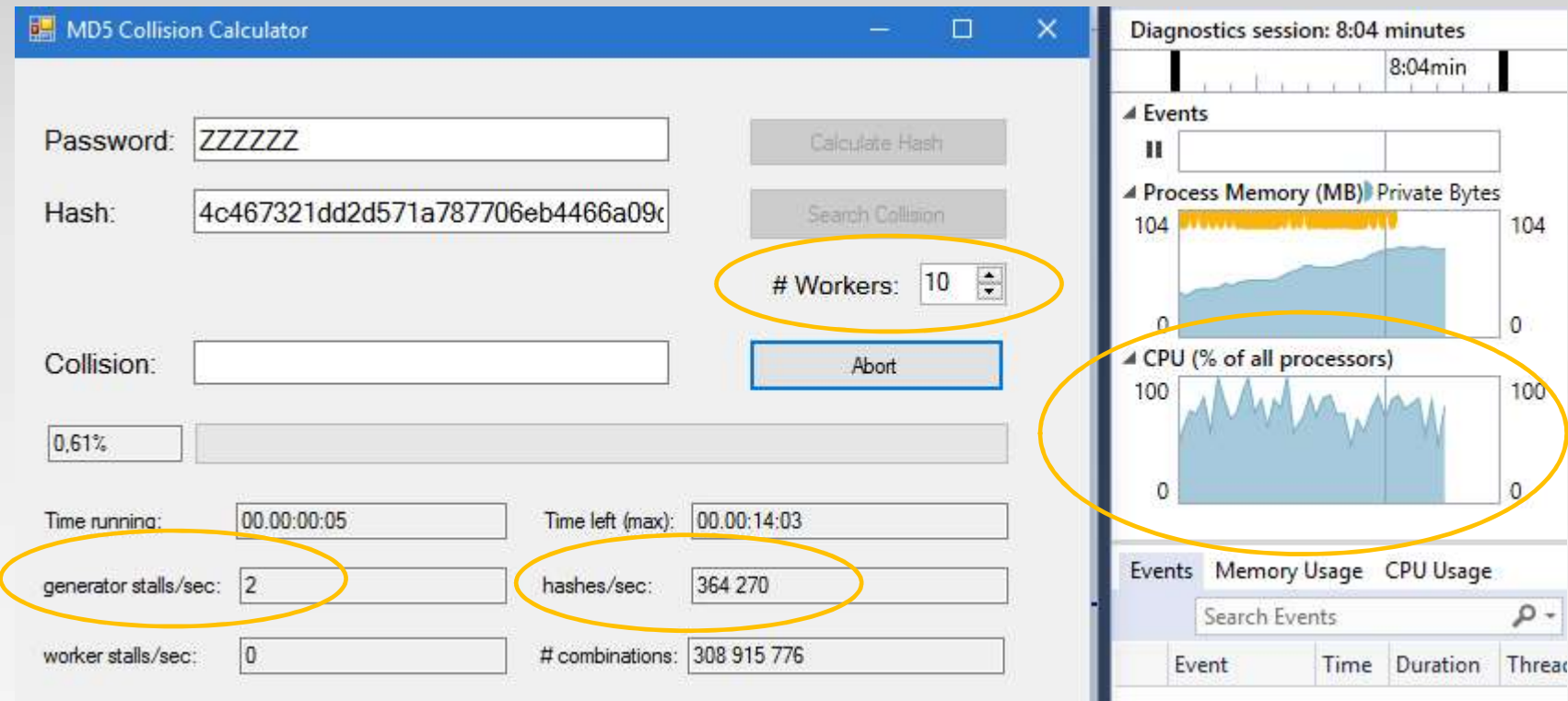
4 workers



Performantie ~ x 3 (iets minder): turboboost uit, competitie met generator, UI, OS...

Tenslotte: profiling (geen stof)

10 workers



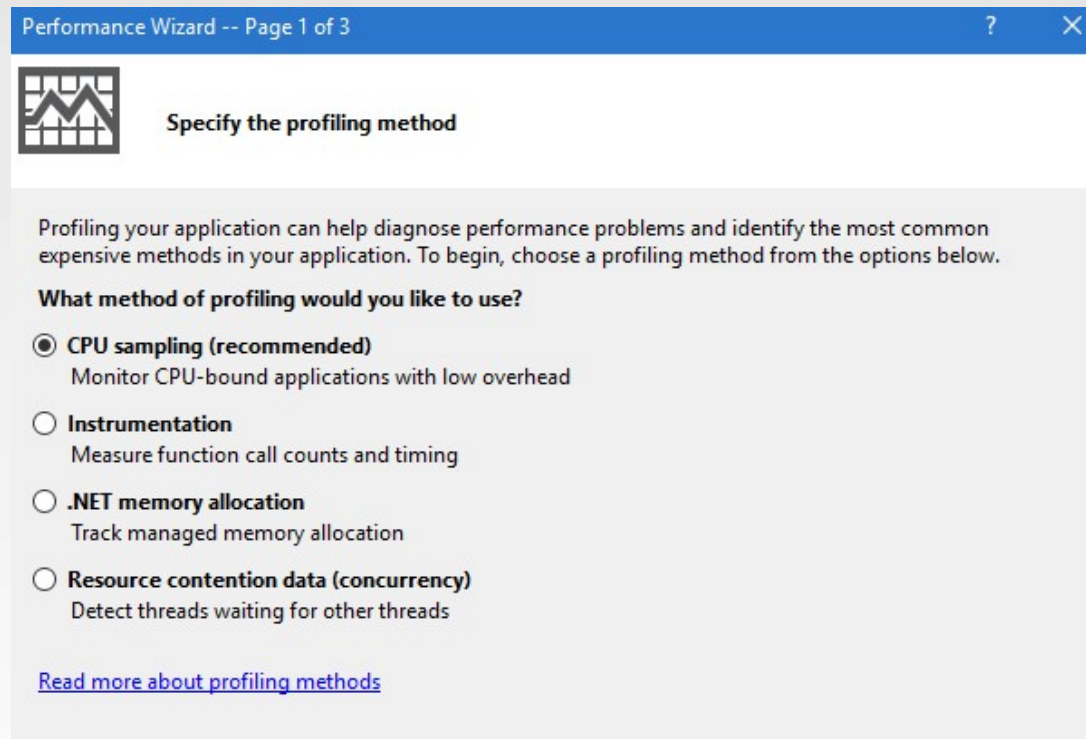
Performantie fractie beter, CPU load ↗, generator stalls ↘ (krijgt minder CPU tijd).
CPU ~ 92%, geen 100% !

Tenslotte: profiling (geen stof)

Profiling: gedetailleerde analyse van CPU-verbruik & bottlenecks

‘Debug’ – ‘Profiler’ – ‘Start diagnostic tools without debugging’

⇒ Performance wizard

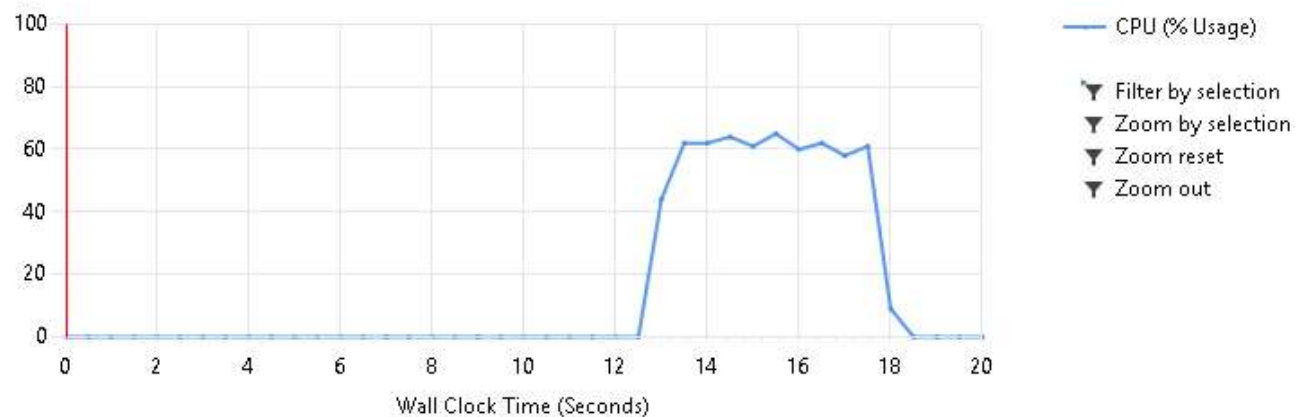


Tenslotte: profiling (geen stof)

Profiling: CPU

Sample Profiling Report

24,352 total samples collected



Hot Path

Function Name	Inclusive Samples %	Exclusive Samples %
MD5CollisionCalculatorGUI.exe	100,00	0,00
BackendImplementation.Worker+<>c__DisplayClass12_0.<GetCollisions>b_3	84,79	0,00
BackendImplementation.Worker.Work	84,67	0,33
GlobalTools.MD5Calculator.GetHash	49,33	1,25
[mscorlib.ni.dll]	37,57	37,57

Tenslotte: profiling (geen stof)

Profiling: CPU

Current View: Modules				
Name	Inclusive Samples	Exclusive Samples	Inclusive Samples %	Exclusive Samples %
mscorlib.ni.dll	20 254	20 254	83,17	83,17
clr.dll	2 926	2 926	12,02	12,02
GlobalTools.dll	12 018	309	49,35	1,27
System.Core.ni.dll	303	303	1,24	1,24
BackendInterface.dll	485	282	1,99	1,16
BackendImplementation.dll	24 237	146	99,53	0,60
System.Windows.Forms.ni.dll	138	122	0,57	0,50
mscorlib.dll	8	8	0,03	0,03
WiFiGO_HookKey.dll	1	1	0,00	0,00
windows.storage.dll	1	1	0,00	0,00
LogicImplementation.dll	31	0	0,13	0,00
lhc_help32-106568.dll	1	0	0,00	0,00
MD5CollisionCalculatorGUI.exe	138	0	0,57	0,00



Conclusie: grootste werk zit in .Net calls opgeroepen vanuit 'GetHash'

Tenslotte: profiling (geen stof)

Conclusie: grootste werk zit in .Net calls opgeroepen vanuit 'GetHash'

= statische methode om MD5-hash te berekenen (via .Net calls)

```
public static string GetHash(string text)
{
    using (var md5 = MD5.Create())
    {
        byte[] data = md5.ComputeHash(Encoding.UTF8.GetBytes(text));

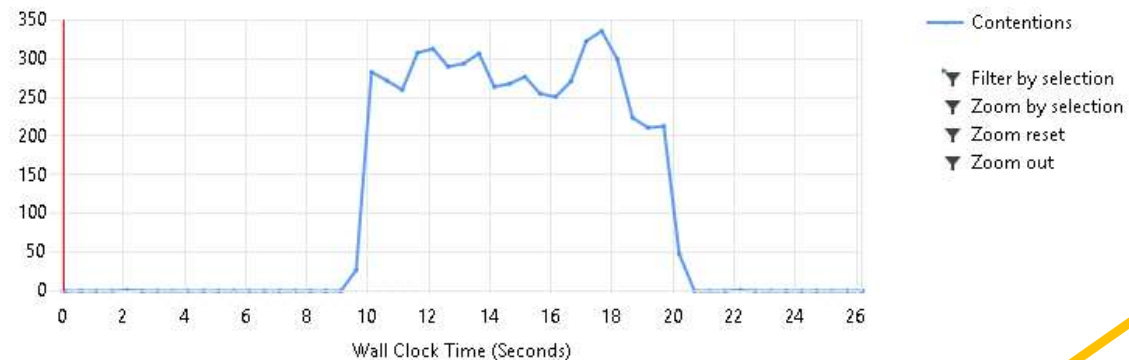
        StringBuilder sBuilder = new StringBuilder();
        for (int i = 0; i < data.Length; i++)
        {
            sBuilder.Append(data[i].ToString("x2"));
        }
        return sBuilder.ToString();
    }
}
```

Tenslotte: profiling (geen stof)

Profiling: resources, concurrency (met 8 workers)

Concurrency Profiling Report

5,004 total contentions



bottleneck !

Most Contended Resources

Name	Contentions %	Contentions
Handle 2	93,21	4 664
Handle 3	6,24	312
Handle 1	0,56	28

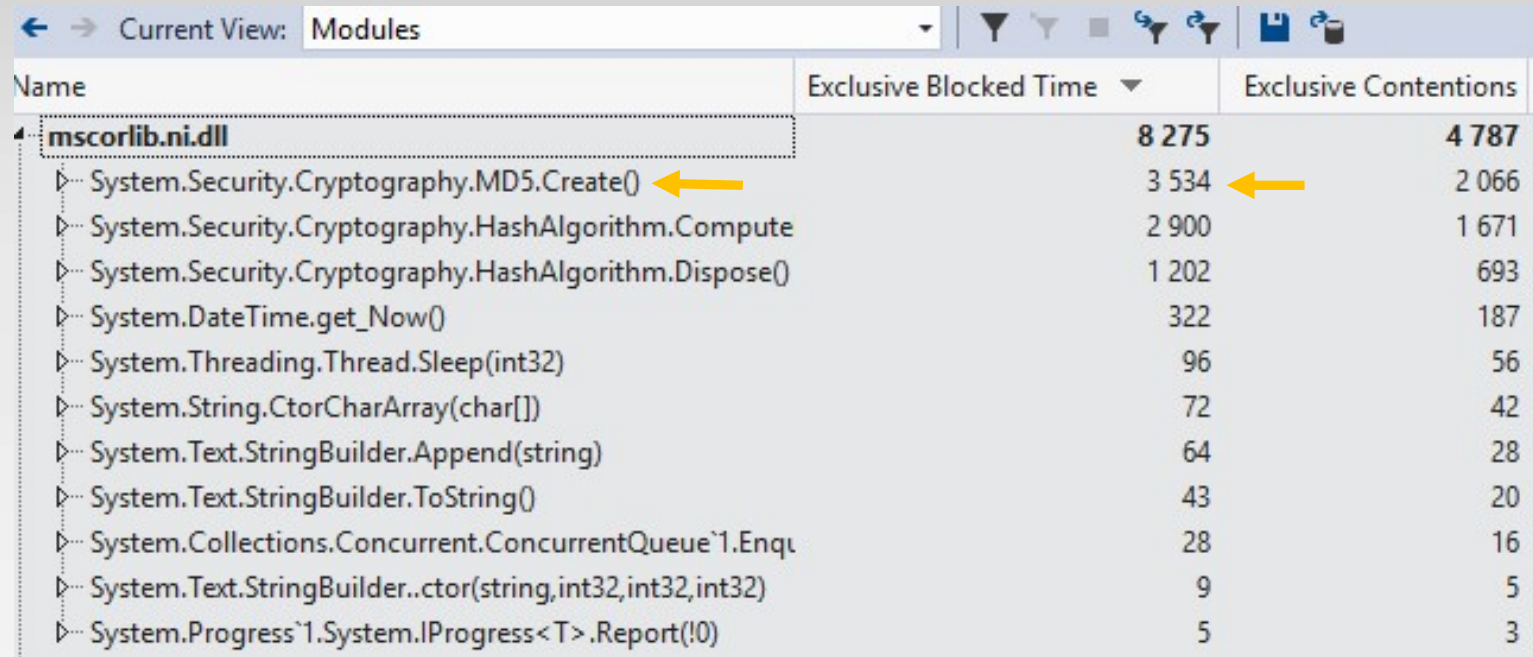
Most Contended Threads

ID	Name	Contentions %	Contentions
18104	[clr.dll]	11,79	590
16560	[clr.dll]	11,71	586
9604	[clr.dll]	11,71	586
14752	[clr.dll]	11,67	584
17844	[clr.dll]	11,57	579

5 threads

Tenslotte: profiling (geen stof)

Profiling: resources, concurrency (met 8 workers) - details



Name	Exclusive Blocked Time	Exclusive Contentions
mscorlib.ni.dll	8 275	4 787
System.Security.Cryptography.MD5.Create()	3 534	2 066
System.Security.Cryptography.HashAlgorithm.Compute	2 900	1 671
System.Security.Cryptography.HashAlgorithm.Dispose()	1 202	693
System.DateTime.get_Now()	322	187
System.Threading.Thread.Sleep(int32)	96	56
System.String.CtorCharArray(char[])	72	42
System.Text.StringBuilder.Append(string)	64	28
System.Text.StringBuilder.ToString()	43	20
System.Collections.Concurrent.ConcurrentQueue`1.Enqueue	28	16
System.Text.StringBuilder..ctor(string,int32,int32,int32)	9	5
System.Progress`1.System.IProgress<T>.Report(!0)	5	3

Conclusie: MD5.Create, Compute, Dispose gebruiken 'lock' mechanisme

⇒ Tasks wachten op elkaar (vandaar geen CPU tijd naar 100%)

Tenslotte: profiling (geen stof)

Profiling:

Conclusie: berekenen van Hash is bottleneck (zowel CPU als lock)

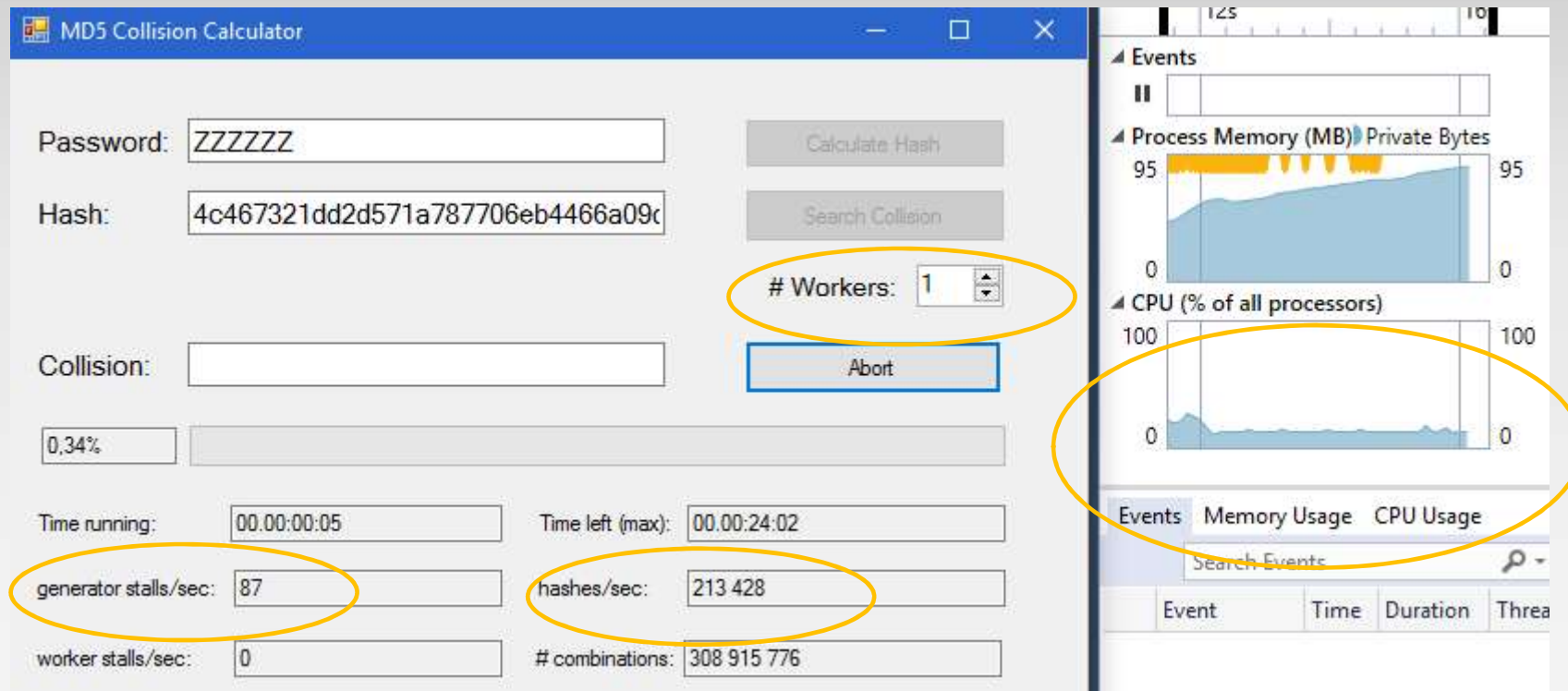
Oplossingen

- Niet met één statische methode werken, maar elke worker zijn eigen versie van MD5 (in een niet-statische methode): éénmalige create doen van MD5 per worker.
- Nog verder gaan: geen .Net calls meer gebruiken voor md5 berekening (die gebruiken impliciet een lock om thread safe te zijn) maar eigen implementatie met eigen lokale variabelen per worker (dus geen lock meer nodig) ⇒ veel werk!

⇒ Enkel 1^e oplossing getest (weinig werk 😊). Resultaten: zie volgende slides...

Tenslotte: profiling (geen stof)

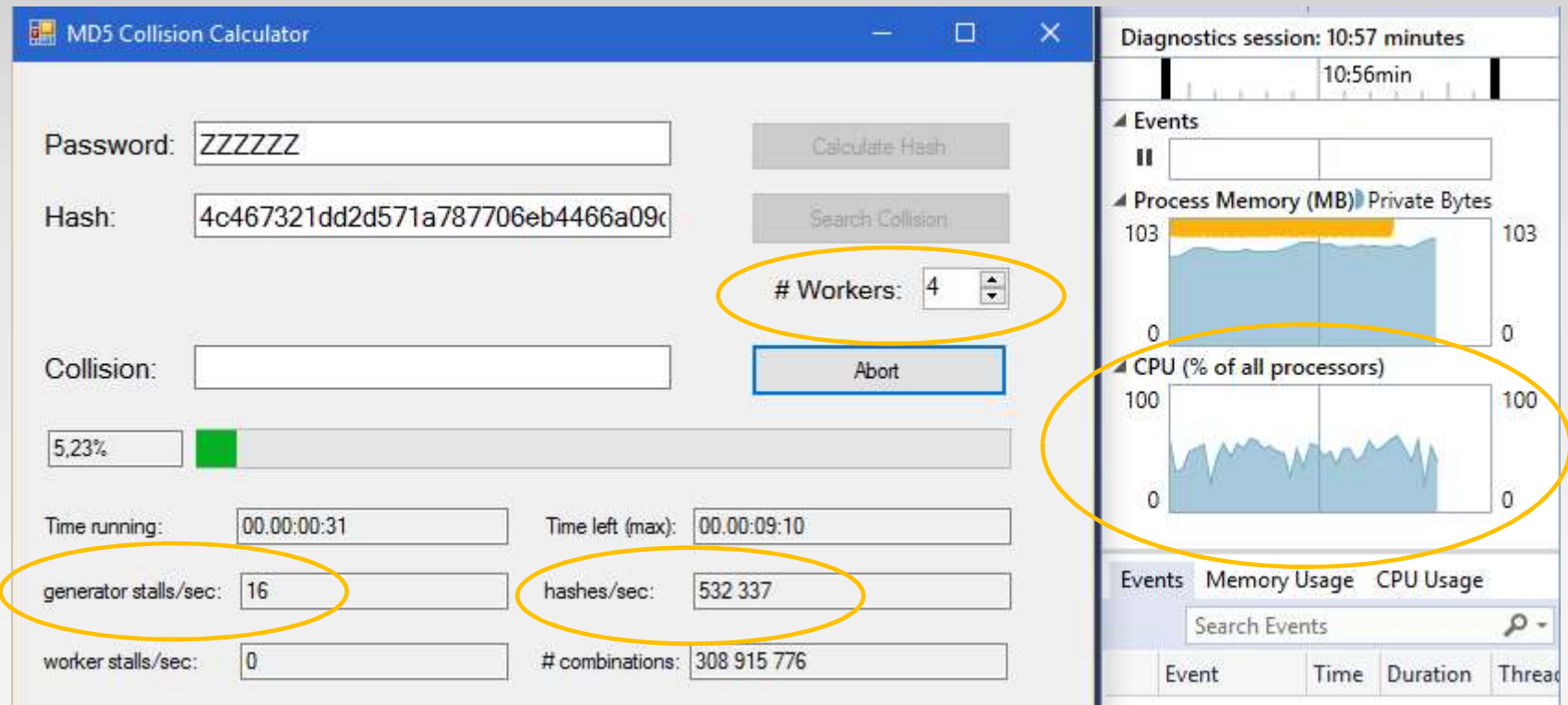
1 worker



Resultaat: 213 k hashes/s ipv 132k (+61%) , minder generator stalls, zelfde CPU-belasting

Tenslotte: profiling (geen stof)

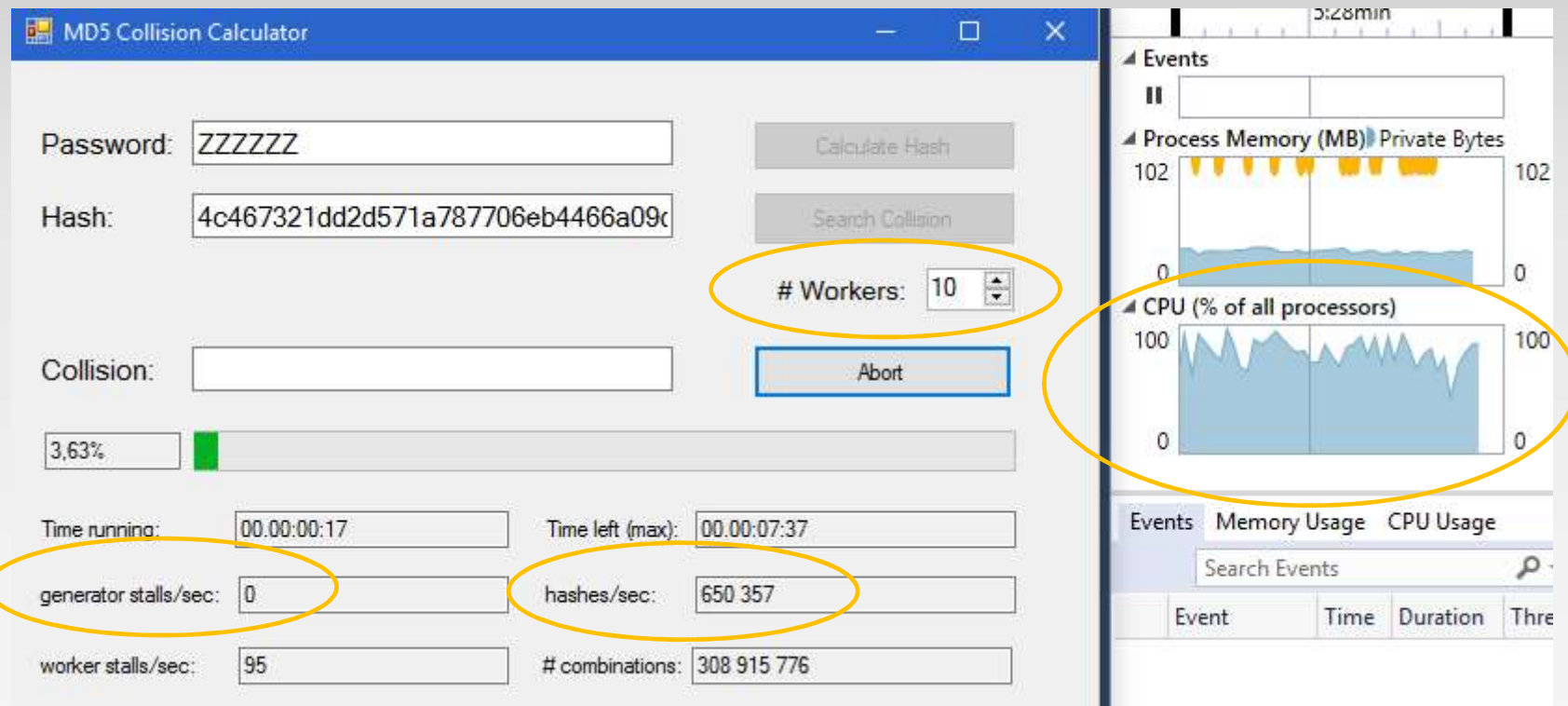
4 workers



Resultaat: 532k hashes/s ipv 336k (+58%), minder generator stalls, zelfde CPU-belasting

Tenslotte: profiling (geen stof)

10 workers



Resultaat: 650k hashes/s ipv 364k (+79%) , geen generator stalls, zelfde CPU-belasting (worker stalls !!!)

