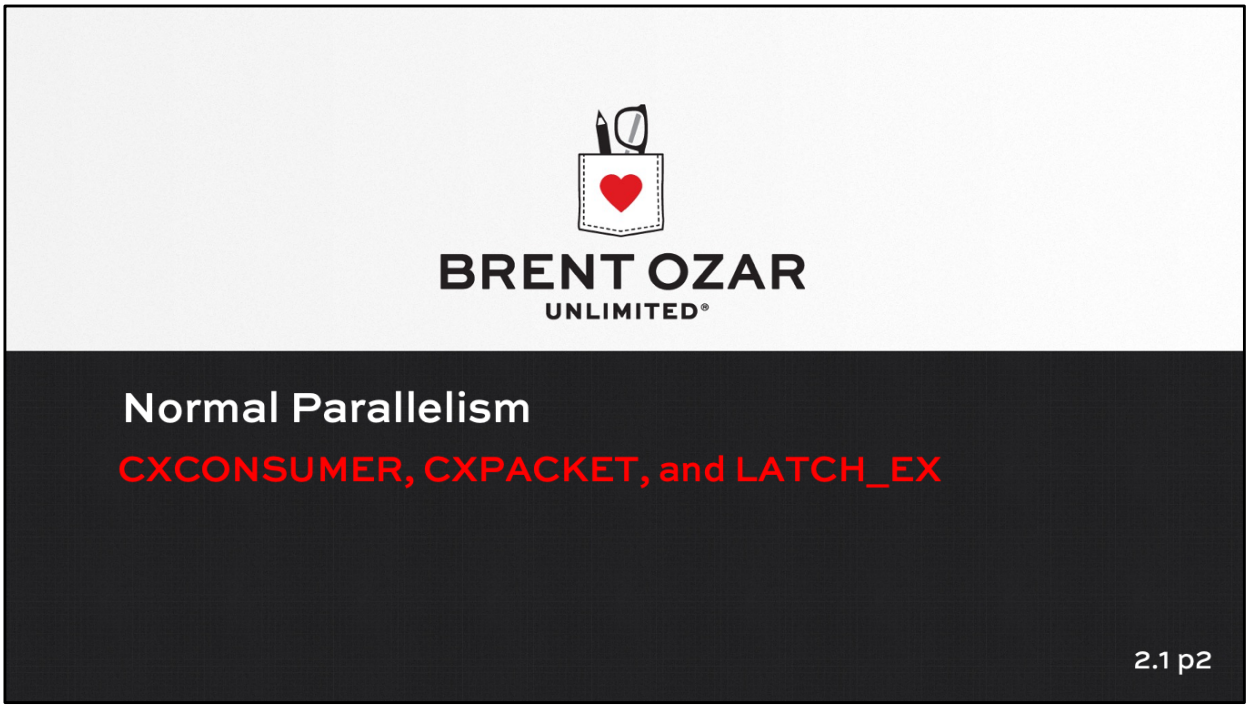
Setting up for the lab

1. **Restart the SQL Server service (clears stats)**
2. **Restore your StackOverflow database**
3. **Copy & run the setup script, will take 1-2 minutes: BrentOzar.com/go/serverlab3**
4. **Start SQLQueryStress:**
    1. File Explorer, D:\Labs, run SQLQueryStress.exe
    2. Click File, Open, D:\Labs\ServerLab3.json
    3. Click Go

2.1 p1

© 2020 Brent Ozar Unlimited®. All rights reserved.



© 2020 Brent Ozar Unlimited®. All rights reserved.

We covered CPU & storage before this.

Queries go parallel because they:

- Need to read a lot of 8KB pages (PAGEIOLATCH)
- Need to do a lot of CPU work (SOS_SCHEDULER_YIELD)

So generally, when I see CXPACKET/CXCONSUMER as the top wait:

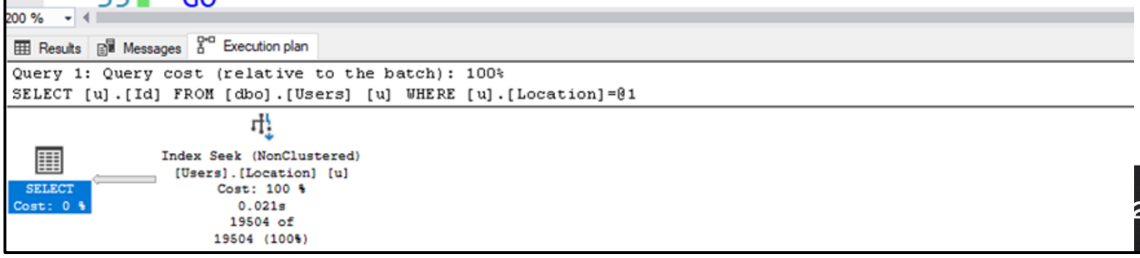
1. Make sure CTFP and MAXDOP have sane defaults (and we'll talk about those)
2. Look past CX*, and look at the next top wait type (typically SOS or PAGEIOLATCH)
3. Tune that, and the parallelism usually falls away
4. But sometimes it doesn't (and we cover that in MQT)

2.1 p3

© 2020 Brent Ozar Unlimited®. All rights reserved.

Easy query plans only use 1 CPU core.

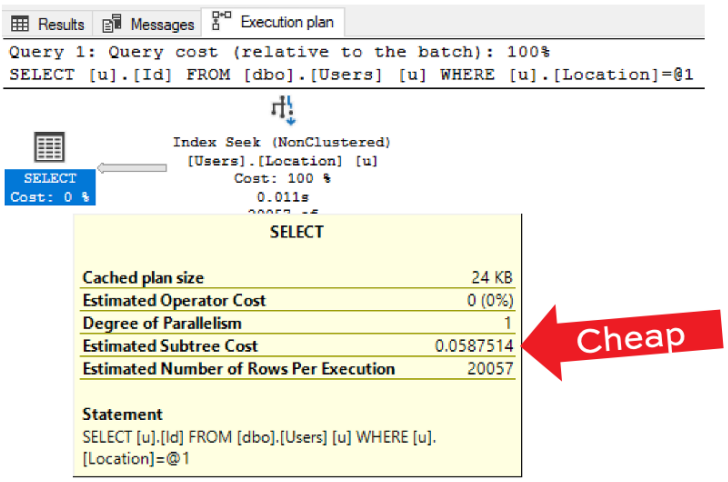
```
28  CREATE INDEX Location ON dbo.Users(Location);
29  GO
30
31  /* If I need to find all the users in one location: */
32  SELECT u.Id
33    FROM dbo.Users u
34    WHERE u.Location = N'London, United Kingdom';
35  GO
```



2.1 p4

© Brent Ozar Unlimited®. All rights reserved.

Because their cost is low.



2.1 p5

© 2020 Brent Ozar Unlimited®. All rights reserved.

Each operator has a CPU and IO cost

And every operator's cost is added together to get the Estimated Subtree Cost of the entire query.

2.1 p6

© 2020 Brent Ozar Unlimited®. All rights reserved.

How query costs were calculated: BrentOzar.com/go/nick

I want to share with you a story I heard from Lubor Kollar who was the Program Manager for the Query Optimizer Team of SQL Server ,and now he is leading the SQL Server Customer Advisory Team .(http://blogs.msdn.com/sqlcat/default.aspx) which is part of the product group

the story goes that when the new query optimizer was developed for SQL server 7.0 in the Query Optimizer team there was a programmer called nick (I am sorry but I do not know his last name) ,he was responsible for calculating query costs (among other things...) ,and he had to decide how to generate the cost number for a query ,so he decided that if query runs for 1 second on his own pc the cost will be ..... 1 ,so we can finally answer the question what is "estimated subtree cost = 1" means,it means ladies and gentleman that it runs for 1 second on nick's machine!

: sometime ago Lubor Kollar sent me a picture of nick's machine ,and here it is

.....rumor's are saying that this pc is now stored in Lubor Kollar's garage

2.1 p7

© 2020 Brent Ozar Unlimited®. All rights reserved.

Kendra Little coined the term “Query Bucks”

BrentOzar.com/go/querybucks

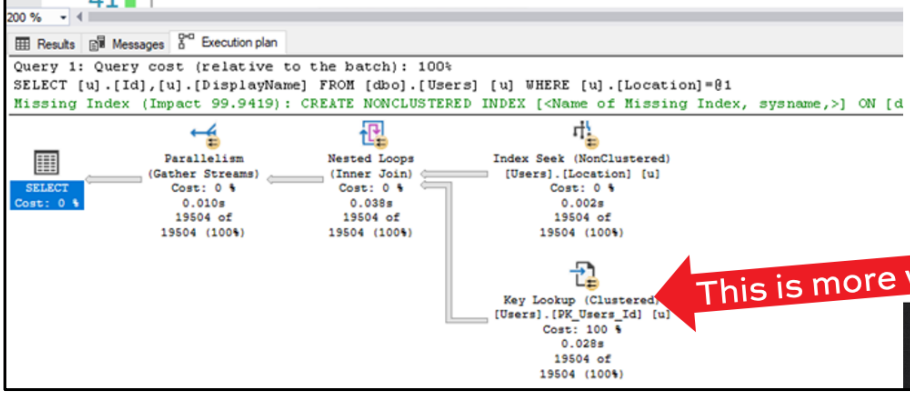
2.1 p8

© 2020 Brent Ozar Unlimited®. All rights reserved.

Add a key lookup, it goes parallel...

```
38  SELECT u.Id, u.DisplayName
39    FROM dbo.Users u
40    WHERE u.Location = N'London, United Kingdom';
41
```

Not in the index

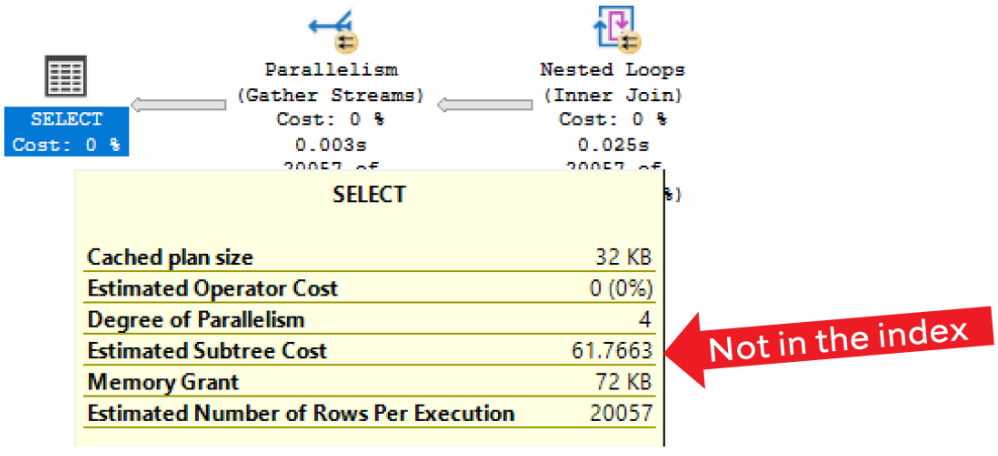


This is more work

2.1 p9

© Brent Ozar Unlimited®. All rights reserved.

Because the query cost went up.

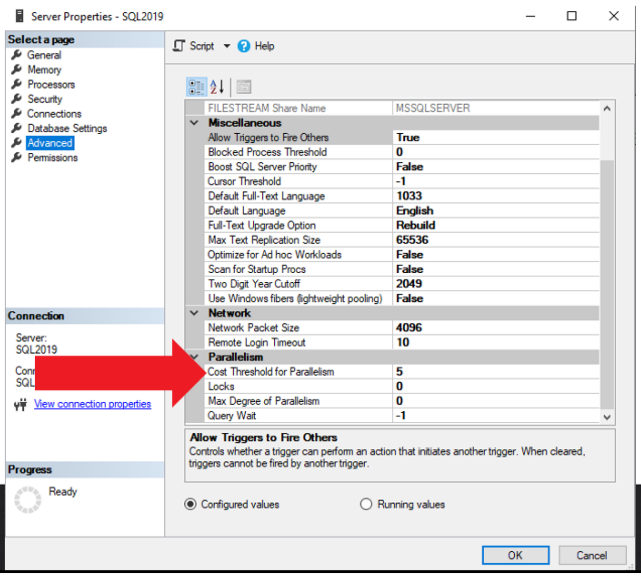


Not in the index

2.1 p10

© 2020 Brent Ozar Unlimited®. All rights reserved.

SQL Server compares the query's total cost to



Server-level setting:
Cost Threshold for Parallelism

Means queries must be this high to ride the parallelism train

2.1 p11

© 2020 Brent Ozar Unlimited®. All rights reserved.

And as long as our query doesn't have these:

- Scalar/MSTVFs/Table Variables
  - Computed columns, check constraints

→ **Far and away the worst, additional overhead**

- TOP
- Global aggregates
- Sequence operators
- Backwards scans

→ Can cause serial zones in plans.

- Recursive CTEs
- Some system tables and functions
- CLR that performs data access
- Dynamic cursors
- OUTPUT clause

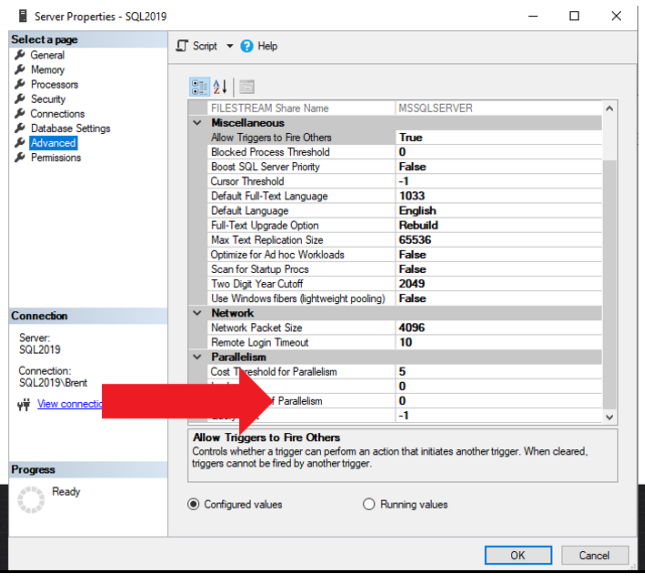
→ Far less common. System functions and views are less likely to cause overall query performance issues.

Also setting MAXDOP to 1

2.1 p12

© 2020 Brent Ozar Unlimited®. All rights reserved.

Then consider parallelizing it at MAXDOP cores.



Up to the number of cores set by Max Degree of Parallelism.

0 = unlimited, as in all of the server's CPU cores.

(There's some other math involved here, but I'm keeping it simple.)

2.1 p13

© 2020 Brent Ozar Unlimited®. All rights reserved.

What MAXDOP controls

MAXDOP controls the maximum number of

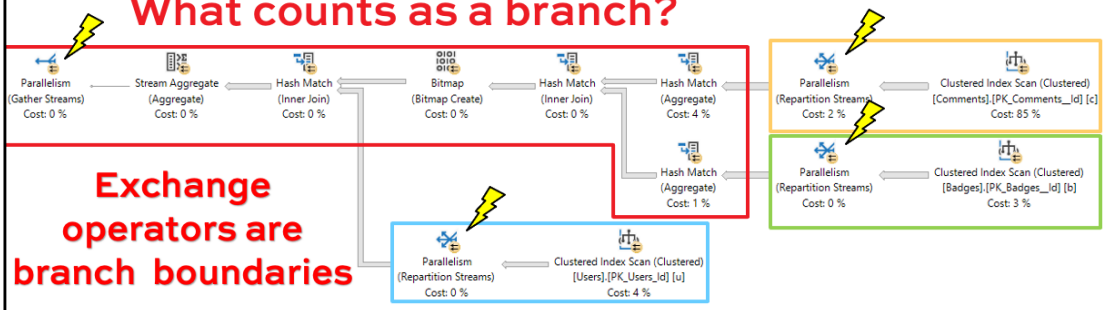
- Cores a query can use
- Threads that a parallel branch can use
- Operators in a branch may share threads

This setting does not control

- The total number of threads a parallel query can use
- The number of threads per parallel operator

© 2020 Brent Ozar Unlimited®. All rights reserved.

What counts as a branch?



Exchange operators are branch boundaries

ThreadStat	
Branches	2
⊞ ThreadReservation	
UsedThreads	8

This is the number of branches that can be active concurrently, not the total branches.

Not all branches can execute concurrently because of blocking operators (hashes, sorts)

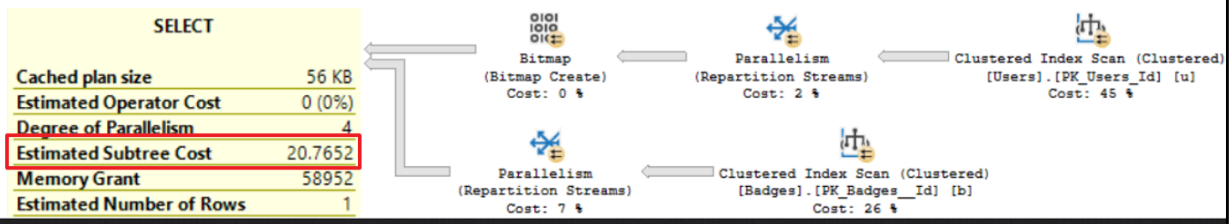
2.1 p15

© 2020 Brent Ozar Unlimited®. All rights reserved.

The parallel plan's cost might be < CTFP

Serial plan cost is what needs to exceed Cost Threshold

To see what the serial plan's cost was,
run it with an OPTION (MAXDOP 1) hint

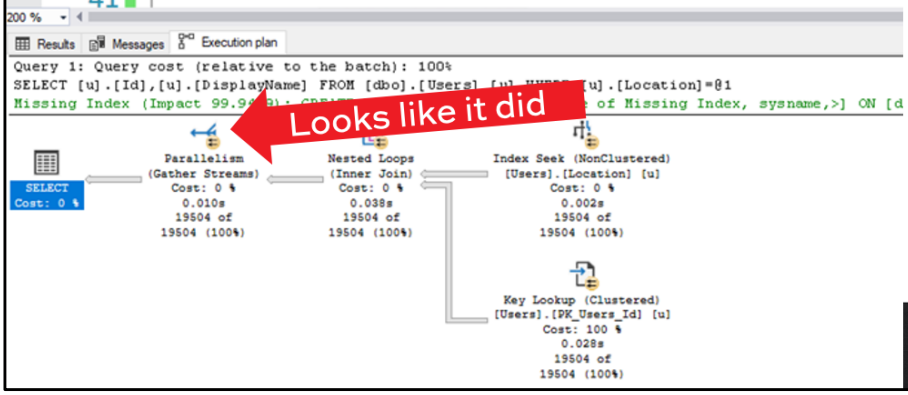


2.1 p16

© 2020 Brent Ozar Unlimited®. All rights reserved.

Back to our query. It went parallel, right?

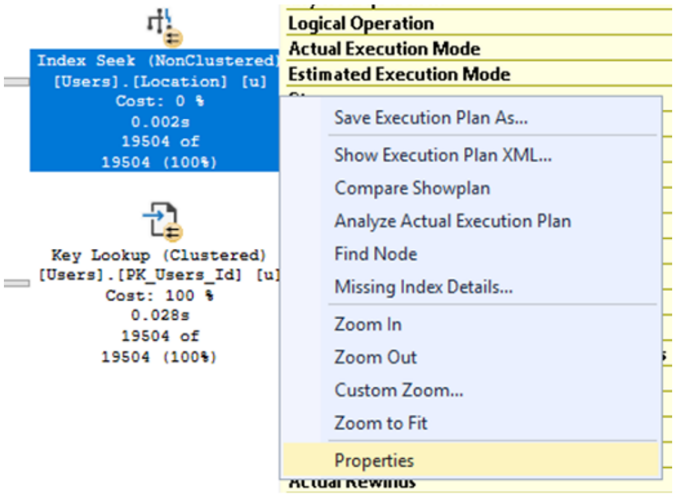
```
38  SELECT u.Id, u.DisplayName
39    FROM dbo.Users u
40    WHERE u.Location = N'London, United Kingdom';
41
```



2.1 p17

© Brent Ozar Unlimited®. All rights reserved.

Well, examine the index seek's properties...



© Brent Ozar Unlimited®. All rights reserved.

And it's not really parallel here.

Thread 0 = coordinating thread

- Reads a little to figure out how to pass out work to the other threads
- He's not actually handling rows though

Thread 1 = doing some work

The rest = bored out of their gourds

2.1 p19

© Brent Ozar Unlimited®. All rights reserved.

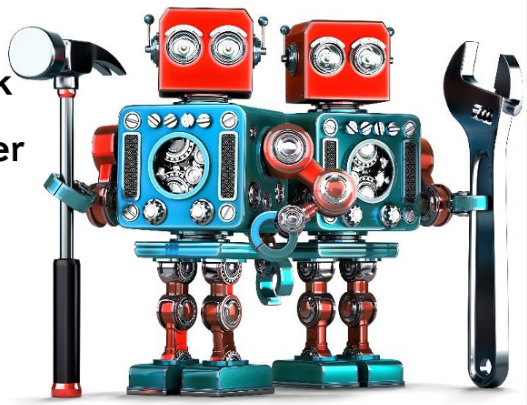
When the robots are working right

Work is divided across threads

Each thread performs equal work

Seamless results returned to user

Each thread needs – and gets – exactly the same amount of memory and CPU time

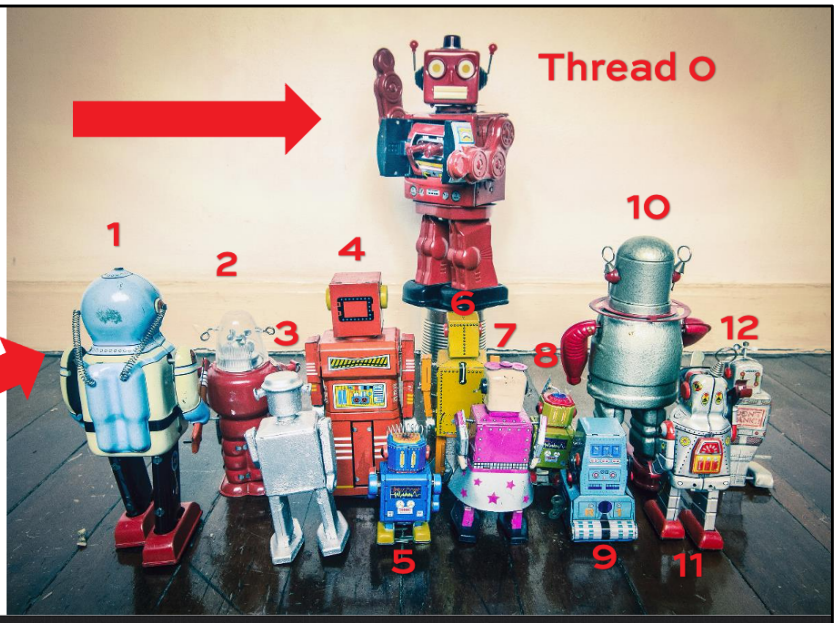


2.1 p20

© 2020 Brent Ozar Unlimited®. All rights reserved.

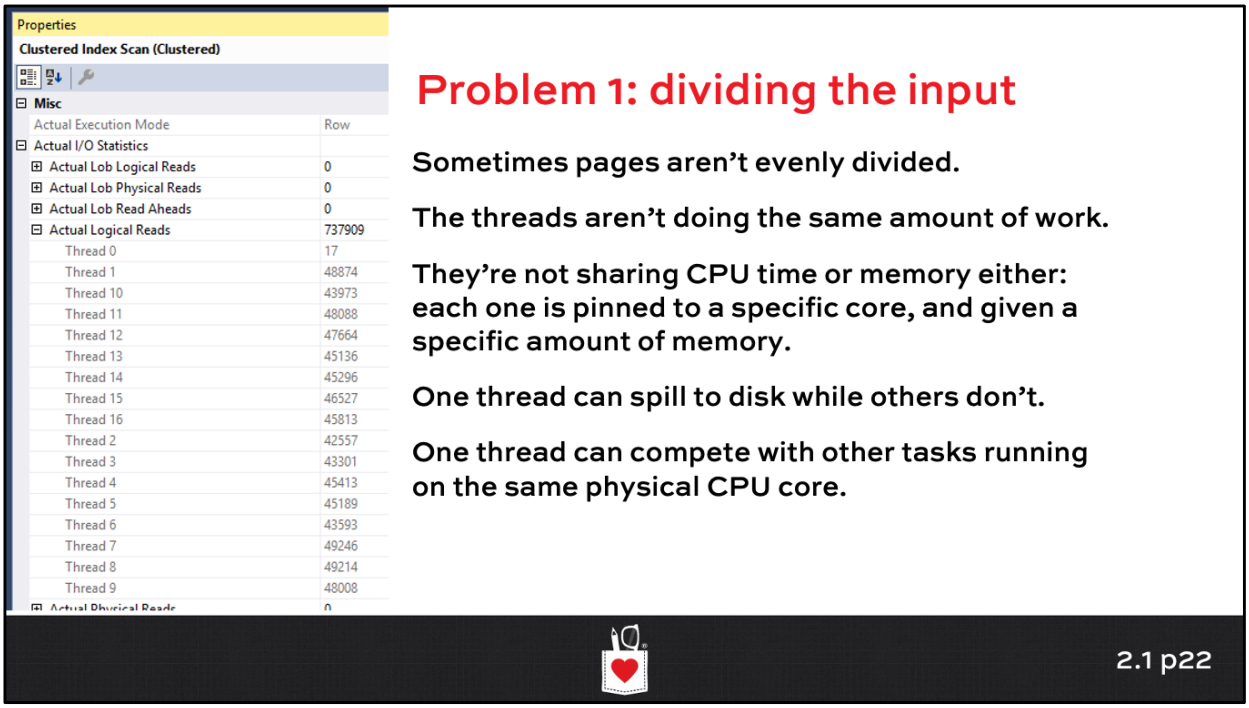
There's a Robot Overlord who tells the other robots to go get rows

There are Robot Workers who obey the Robot Overlord



2.1 p21

© 2020 Brent Ozar Unlimited®. All rights reserved.



© 2020 Brent Ozar Unlimited®. All rights reserved.

Problem 2: unpredictable output

We simply don't know which threads will find results.

For example, if we don't have an index
and we have to do a table scan to find matches,
we just don't know which threads will find matches.

The matches could be anywhere in the table.

One thread might just find all of them.

2.1 p23

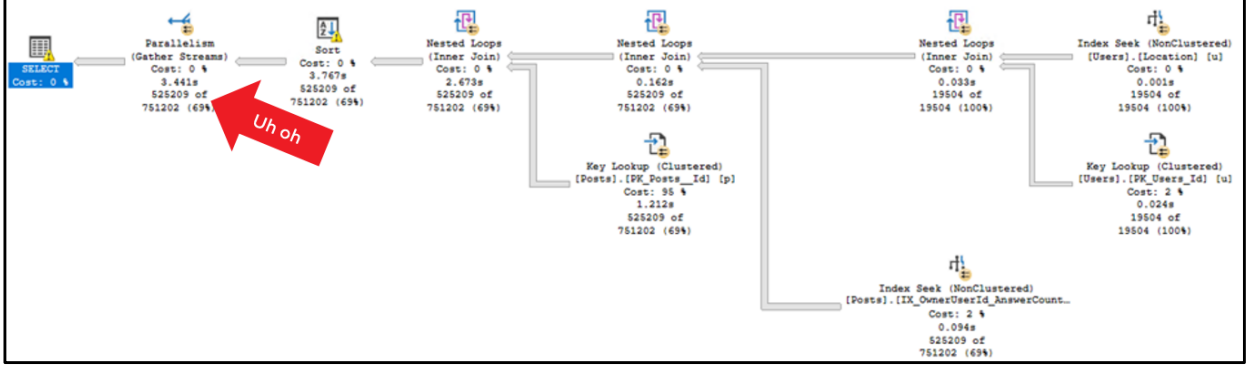
© 2020 Brent Ozar Unlimited®. All rights reserved.

When unpredictable output is bad

By itself – just reading out rows – it's not a big deal.

But what if *several* operators are daisy-chained together in the same parallel operation, like we cover in Mastering Query Tuning?

One thread may do all of the work, period.



© 2020 Brent Ozar Unlimited®. All rights reserved.

Problem 3: odd parallelism bugs

I call these bugs, but Microsoft disagrees:

- Index spools say they're parallel, but they're not
- Order preserving exchanges like parallel merge joins perform terribly

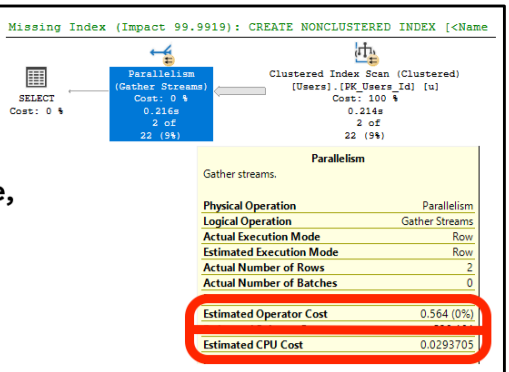
We'll cover these in a separate module on Abnormal Parallelism.

© 2020 Brent Ozar Unlimited®. All rights reserved.

Problem 4: parallelism itself has a cost, too.

This one's cheap, but the more data you have, the higher the cost becomes.

Plus, in bigger plans, we have to rebalance the work as we move through the operators, distributing streams of data differently to different threads to compensate for the fact that we don't know which threads will find work.



© 2020 Brent Ozar Unlimited®. All rights reserved.

How we measure parallelism

Normally, we use wait stats to measure stuff.

Parallelism makes things tricky though:

- Every robot tracks waits
- The overlord tracks waits too

While a parallel query is running,
the robot overlord (thread 0) piles up waits
even though he's not really waiting on things,
and we can't possibly get rid of his "wait" time.

Robot workers also track "wait" time when they're done
with their task, and "waiting" on other robots to finish.

2.1 p27

© 2020 Brent Ozar Unlimited®. All rights reserved.

This is where it gets really terrible.

The robot overlord will always be waiting, and that's totally okay.

The robot workers should NOT be waiting, because that indicates:

- Problem 1: we didn't balance work evenly
- Problem 2: the workers found rows that caused them to do unbalanced work
- Problem 3: there was too much overhead in passing out the data

But for decades, both the robot overlord's wait time and the robot workers' wait time had the same wait type.

2.1 p28

© 2020 Brent Ozar Unlimited®. All rights reserved.

The CXPACKET debacle

Microsoft finally fixed this in 2016 SP2 & 2017 CU3:

	Robot Overlord	Robot Worker
SQL Server 2017 CU3	CXCONSUMER ("OK")	CXPACKET
SQL Server 2017 pre-CU3	CXPACKET	CXPACKET
SQL Server 2016 SP2	CXCONSUMER ("OK")	CXPACKET
SQL Server 2016 pre-SP2	CXPACKET	CXPACKET
SQL Server 2014 & prior	CXPACKET	CXPACKET

CXCONSUMER was added to separate the coordinator thread waiting on child threads to finish, vs. child threads waiting on each other to finish

2.1 p29

© 2020 Brent Ozar Unlimited®. All rights reserved.

But it's still really confusing.

CXCONSUMER *should* be totally ignorable – but it's not.

We've found queries in the wild that *only* produce CXCONSUMER waits during unbalanced parallelism.

CXPACKET *should* mean we have a parallelism problem, but which one?

- Problem 1: we didn't balance work evenly
- Problem 2: the workers found rows that caused them to do unbalanced work
- Problem 3: odd parallelism bugs
- Problem 4: too much overhead passing out the data

You just can't tell from wait stats.

2.1 p30

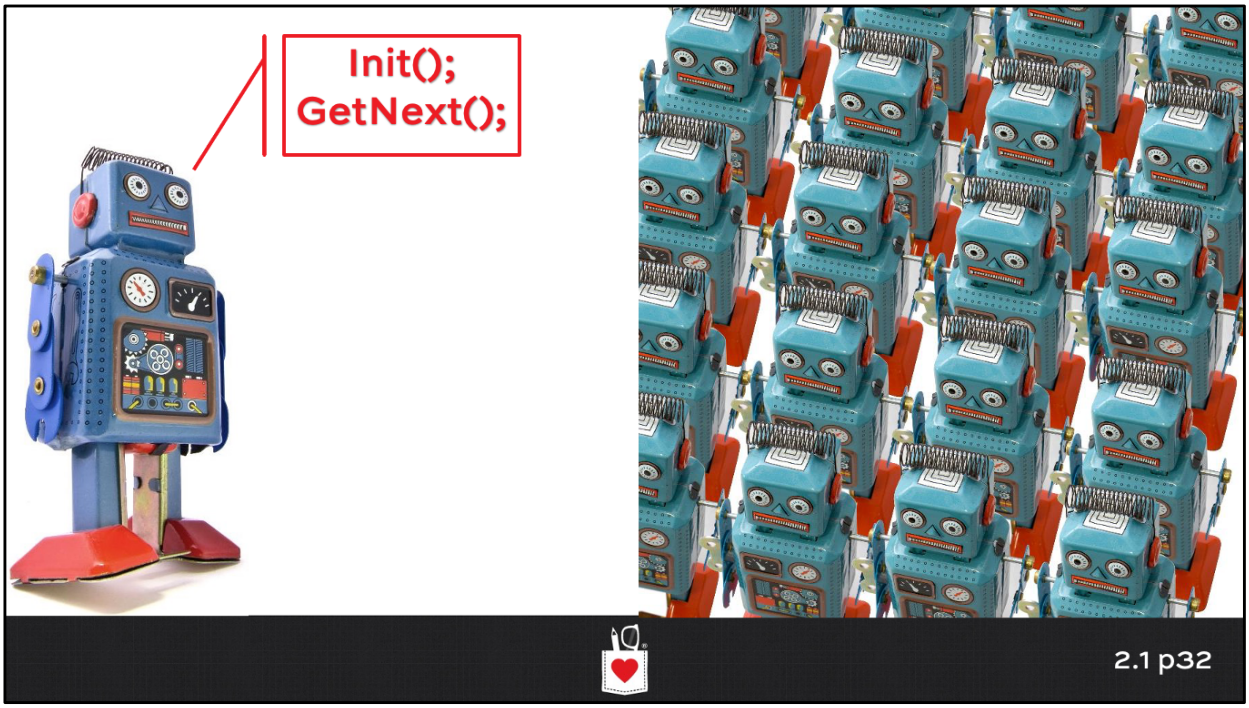
© 2020 Brent Ozar Unlimited®. All rights reserved.

You run a “big” query

```
SELECT COUNT(*) AS records
FROM dbo.Posts AS p
JOIN dbo.Votes AS v
     ON v.PostId = p.Id
JOIN dbo.Comments AS c
     ON c.PostId = v.PostId
JOIN dbo.Users AS u
     ON u.Id = p.OwnerUserId
WHERE v.PostId < 10000000
```

2.1 p31

© 2020 Brent Ozar Unlimited®. All rights reserved.

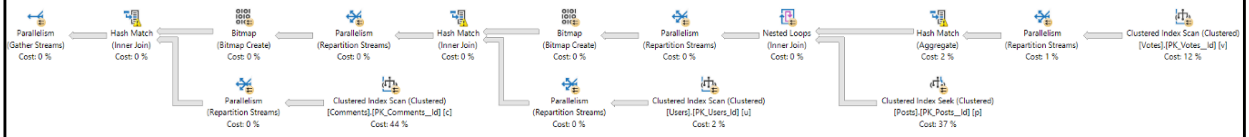


© 2020 Brent Ozar Unlimited®. All rights reserved.

This does a lot of CPU work for ~25 seconds

```
EXEC dbo.sp_BlitzFirst @Seconds = 30, @ExpertMode = 1;
```

Seconds Sample	wait_type	wait_category	Wait Time (Seconds)	Per Core Per Second	Signal Wait Time (Seconds)	Percent Signal Waits	Number of Waits	Avg ms Per Wait
30	CXPACKET	Parallelism	63.2	0.2	29.9	47.3	17861	3.5
30	CXCONSUMER	Parallelism	45.3	0.1	26.3	58.1	24440	1.9
30	SOS_SCHEDULER_YIELD	CPU	16.6	0.0	16.6	100.0	16391	1.0
30	PREEMPTIVE_OS_WRITEFILE	Preemptive	0.1	0.0	0.0	0.0	4	19.0
30	RESERVED_MEMORY_ALLOCATION_EXT	Memory	0.1	0.0	0.0	0.0	208370	0.0
30	SOS_PHYS_PAGE_CACHE	Other	0.0	0.0	0.0	0.0	1	1.0
30	LATCH_EX	Latch	0.0	0.0	0.0	0.0	12	0.1
30	MEMORY_ALLOCATION_EXT	Memory	0.0	0.0	0.0	0.0	4539	0.0



2.1 p33

© 2020 Brent Ozar Unlimited®. All rights reserved.

But is parallelism really the problem?

If you looked at a server with queries like this running all the time, you'd see a whole lot of parallelism waits

We see a lot of servers like this

- They've been up for 10 days
- With 20-30 days of parallelism waits

If we change MAXDOP to 1,
will performance get better?

© 2020 Brent Ozar Unlimited®. All rights reserved.

All the single threadies

```
SELECT COUNT(*) AS records
FROM dbo.Posts AS p
JOIN dbo.Votes AS v
     ON v.PostId = p.Id
JOIN dbo.Comments AS c
     ON c.PostId = v.PostId
JOIN dbo.Users AS u
     ON u.Id = p.OwnerUserId
WHERE v.PostId < 10000000
OPTION(MAXDOP 1)
```

2.1 p35

© 2020 Brent Ozar Unlimited®. All rights reserved.

Not exactly

```
EXEC dbo.sp_BlitzFirst @Seconds = 90, @ExpertMode = 1;
```

Seconds Sample	wait_type	wait_category	Wait Time (Seconds)	Per Core Per Second	Signal Wait Time (Seconds)	Percent Signal Waits	Number of Waits	Avg ms Per Wait
90	PREEMPTIVE_OS_WRITEFILE	Preemptive	0.1	0.0	0.0	0.0	4	15.5
90	RESERVED_MEMORY_ALLOCATION_EXT	Memory	0.1	0.0	0.0	0.0	201816	0.0
90	SOS_SCHEDULER_YIELD	CPU	0.0	0.0	0.0	0.0	21250	0.0
90	MEMORY_ALLOCATION_EXT	Memory	0.0	0.0	0.0	0.0	7524	0.0

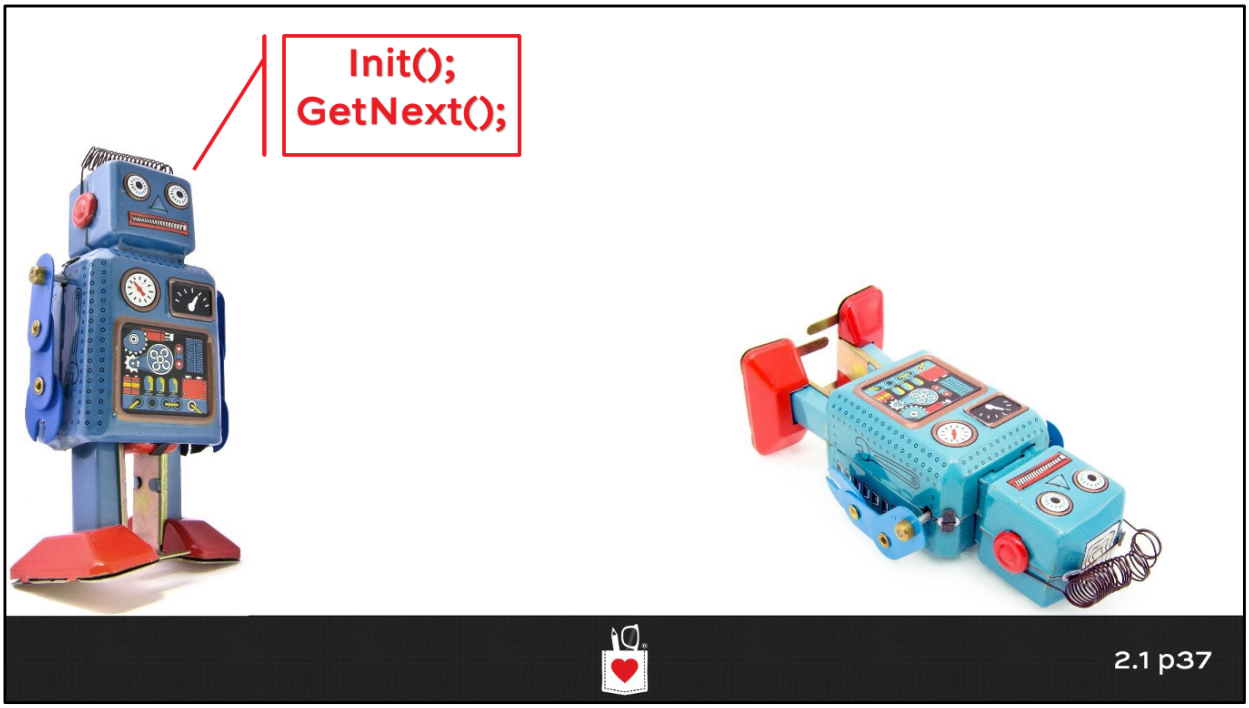
This runs for nearly 90 seconds

It got 3x slower, but we got rid of parallelism

Yay?

2.1 p36

© 2020 Brent Ozar Unlimited®. All rights reserved.



© 2020 Brent Ozar Unlimited®. All rights reserved.

So when you see CXPACKET/CXCONSUMER

1. Set CTFP & MAXDOP per industry best practices.
2. Let those settings bake in for a few days.
3. Make index improvements that can reduce times when queries go parallel to scan lots of data & sort it.
4. Let those settings bake in for a few days.
5. Look for queries still doing a lot of work, then tune them.
6. Let those bake in for a few days.
7. Finally, if parallelism is still a serious problem (like 10X wait time ratio), start looking for specific queries with parallelism issues.

© 2020 Brent Ozar Unlimited®. All rights reserved.

1. Setting Cost Threshold for Parallelism

Default: 5 query bucks

I default to: 50 query bucks

I'm fine with a range of: 30-100

But be wary: the higher you set it,
remember that multi-threaded queries
may suddenly start going single-threaded.

Changing this setting clears the plan cache.

© 2020 Brent Ozar Unlimited®. All rights reserved.

1. Setting Max Degree of Parallelism

Microsoft says: https://support.Microsoft.com/kb/2806535

SQL Server 2016 (13.x) and higher

Starting with SQL Server 2016 (13.x), during service startup if the Database Engine detects more than eight physical cores per NUMA node or socket at startup, soft-NUMA nodes are created automatically by default. The Database Engine takes care of placing logical processors from the same physical core into different soft-NUMA nodes. The recommendations in the table below are aimed at keeping all the worker threads of a parallel query within the same soft-NUMA node. This will improve the performance of the queries and distribution of worker threads across the NUMA nodes for the workload.

Starting with SQL Server 2016 (13.x), use the following guidelines when you configure the **max degree of parallelism** server configuration value:

Server with single NUMA node	Less than or equal to 8 logical processors	Keep MAXDOP at or below # of logical processors
Server with single NUMA node	Greater than 8 logical processors	Keep MAXDOP at 8
Server with multiple NUMA nodes	Less than or equal to 16 logical processors per NUMA node	Keep MAXDOP at or below # of logical processors per NUMA node
Server with multiple NUMA nodes	Greater than 16 logical processors per NUMA node	Keep MAXDOP at half the number of logical processors per NUMA node with a MAX value of 16

2.1 p40

© 2020 Brent Ozar Unlimited®. All rights reserved.

3. Making index improvements to reduce CX%

After letting the parallelism changes bake in for a few days:

sp_BlitzIndex @GetAllDatabases = 1

- Look for high-value missing indexes
- Remember that they're straight from Clippy, though: use the Mastering Index Tuning techniques

sp_BlitzCache @SortOrder = 'reads' and = 'cpu'

- Do manual index tuning to reduce scans, sorts
- Optionally, while you're there: tune the queries

Then let these bake in for a few days too.

© 2020 Brent Ozar Unlimited®. All rights reserved.

5. Look for queries still doing a lot of work

After letting the index changes bake in for a few days:

sp_BlitzCache @SortOrder = 'reads' and = 'cpu'

- Do manual index tuning to reduce scans, sorts
- Optionally, while you're there: tune the queries

And most of the time, that process just obliterates CX% waits.

© 2020 Brent Ozar Unlimited®. All rights reserved.

But sometimes, 1-6 aren't enough because specific queries have abnormal parallelism.

1. Set CTFP & MAXDOP per industry best practices.
2. Let those settings bake in for a few days.
3. Make index improvements that can reduce times when queries go parallel to scan lots of data & sort it.
4. Let those settings bake in for a few days.
5. Look for queries still doing a lot of work, then tune them.
6. Let those bake in for a few days.
7. Finally, if parallelism is still a serious problem (like 10X wait time ratio), start looking for specific queries with parallelism issues.

2.1 p43

© 2020 Brent Ozar Unlimited®. All rights reserved.

Recap

2.1 p44

© 2020 Brent Ozar Unlimited®. All rights reserved.

Start with the basics.

Queries go parallel because they:

- Need to read a lot of 8KB pages (PAGEIOLATCH, LATCH_EX)
- Need to do a lot of CPU work (SOS_SCHEDULER_YIELD)

So when I see CXPACKET/CXCONSUMER as the top waits:

1. Make sure CTFP and MAXDOP have sane defaults (CTFP 30-100, MAXDOP per KB #2806535)
2. Do normal index tuning and query tuning
3. The parallelism usually falls away – but if it doesn't, review the abnormal parallelism causes & fixes in Mastering Query Tuning

© 2020 Brent Ozar Unlimited®. All rights reserved.