BRENT OZAR
UNLIMITED®

Parallelism

CXPACKET, CXCONSUMER, and LATCH_EX

2.1 p1

Parallelism is misunderstood.

It's NOT about
evenly dividing work across CPU cores.

It's about
making queries faster than if they ran on only 1 CPU core.

So when you see parallelism waits

1. Make sure CTFP & MAXDOP have sane defaults
2. Look past parallelism waits, check your next top wait (likely SOS or PAGEIOLATCH)
3. Tune those queries/indexes, and then queries won't need to go *so far* parallel, and parallelism waits will drop away
4. Where it doesn't, revisit Mastering Query Tuning

Agenda

How queries go parallel

Why they can't go evenly parallel

Why parallelism wait times aren't accurate

How to set CTFP & MAXDOP

What SQL Server 2022 is doing to fix it

How queries go parallel

Flash back to Query Tuning

Remember the choose-your-own-adventure book?

When SQL Server compiles queries:

- It builds a plan
- It decides whether it makes sense to build another plan, or just go execute this plan now

Easy plans are considered first, harder ones later

SQL starts single-threaded.

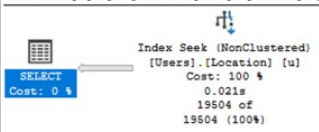
Easy query plans only use 1 CPU core.

```
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
```

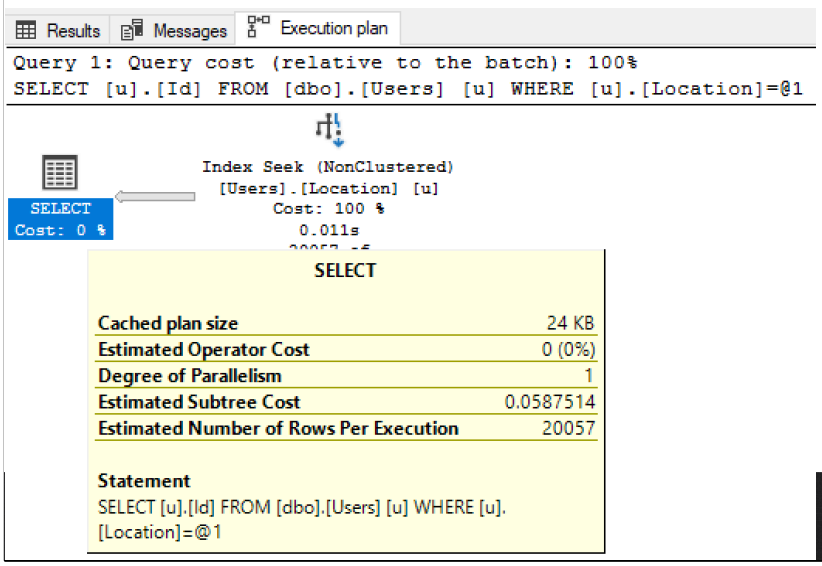
200 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT [u].[Id] FROM [dbo].[Users] [u] WHERE [u].[Location]=@1
```

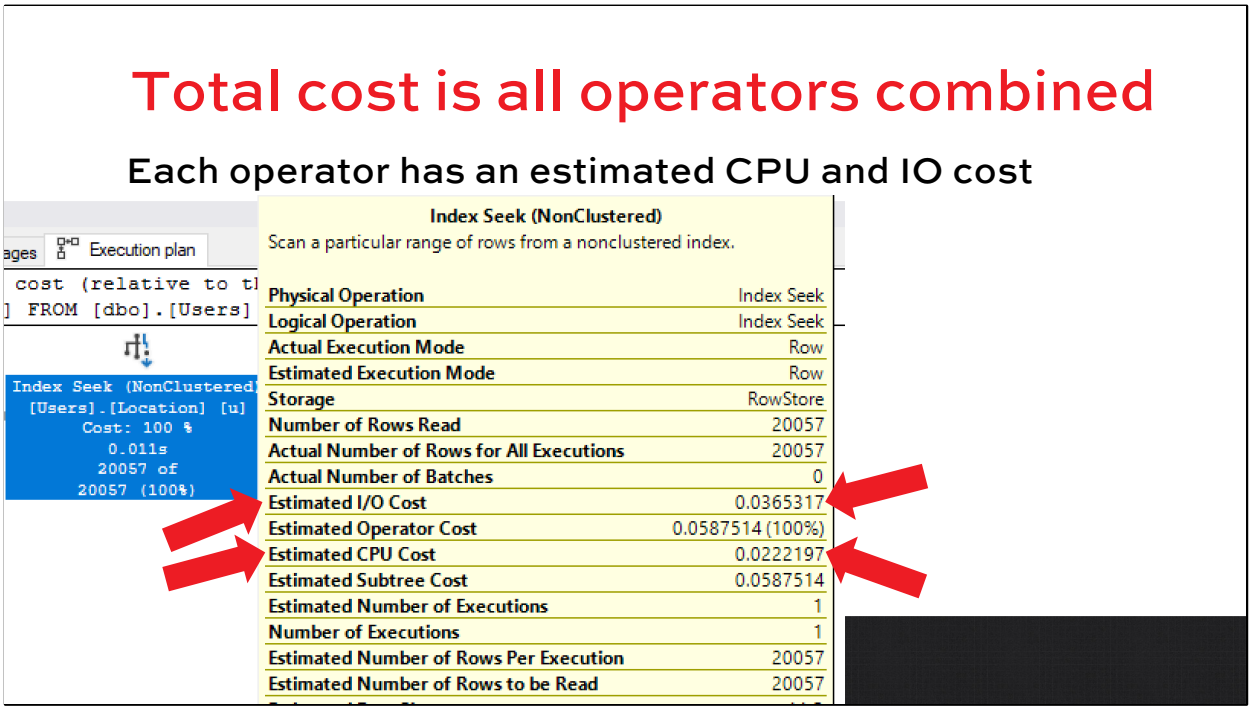


The plan's cost is cheap.



Total cost is all operators combined

Each operator has an estimated CPU and IO cost



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

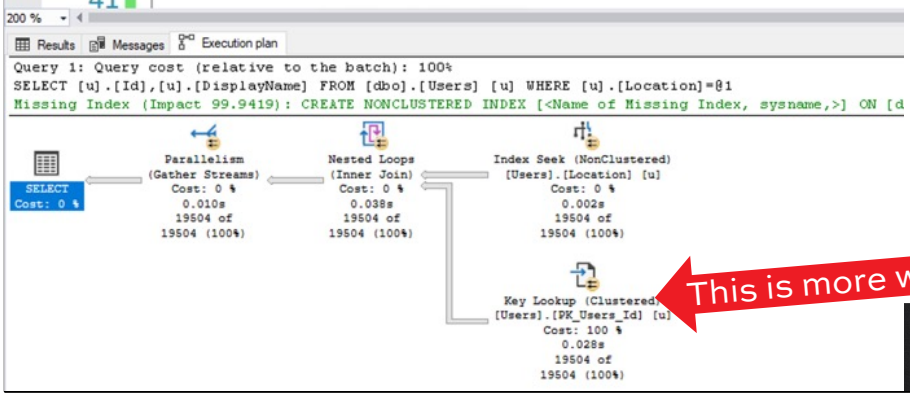
.....rumor's are saying that this pc is now stored in Lubor Kollar's garage

BrentOzar.com/go/querybucks

Add a key lookup, it goes parallel...

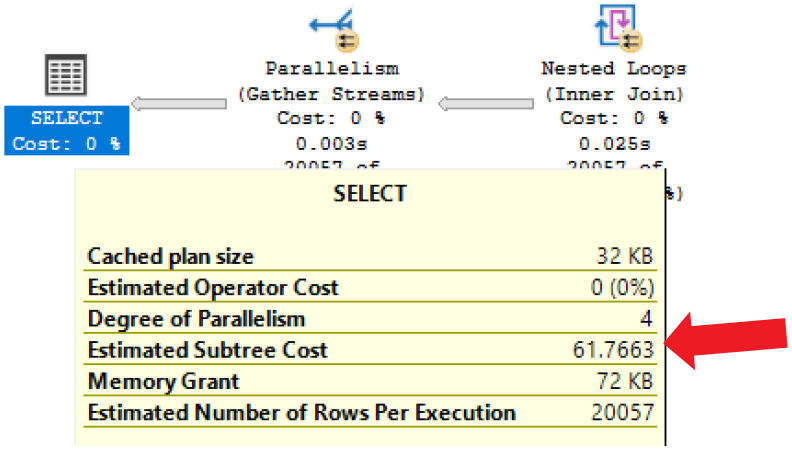
```
38  SELECT u.Id, u.DisplayName
39    FROM dbo.Users u
40    WHERE u.Location = N'London, United Kingdom';
41
```

Not in the index



This is more work

Because query cost is way up.



What happened

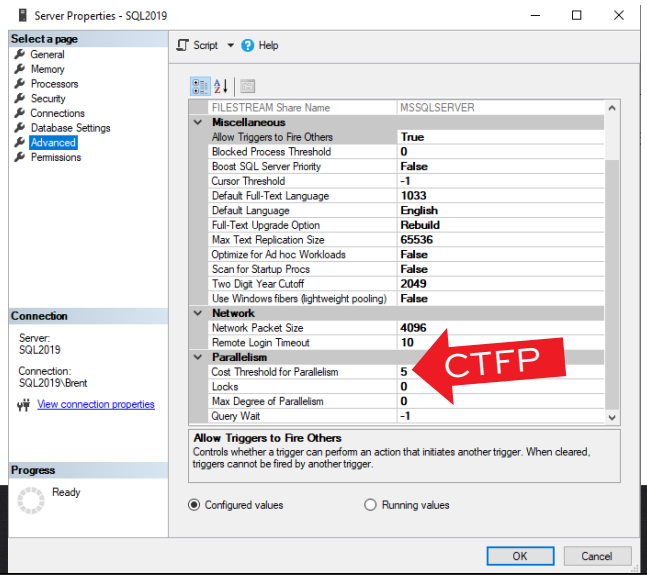
The query optimizer (architect) was going through the choose-your-own-adventure book

The plan he built was relatively expensive

He had to decide whether to take another adventure, this time building a parallel plan

He looked at the query's cost in query bucks and compared it to a server-level setting

Cost Threshold for Parallelism



2.1 p15

He also reviewed the query

Looking for parallelism blockers like:

- Scalar functions
- Table variables
- TOP
- Sequence operators
- Backwards scans
- Recursive CTEs

Full list: BrentOzar.com/go/serialudf

Expensive? No blockers?

Went on another adventure building a parallel plan

The Max Degree of Parallelism (MAXDOP) controls how many CPU cores get used (0 – unlimited)

Parallelism	
Cost Threshold for Parallelism	5
Locks	0
Max Degree of Parallelism	0
Query Wait	-1

More ways to control it: BrentOzar.com/go/maxdop

But MAXDOP isn't max.

It should just be called Degrees of Parallelism.

SQL Server doesn't start small and increase.

It doesn't even gradually back down til 2022.

Queries go parallel at MAXDOP cores except in rare cases like extreme memory pressure.

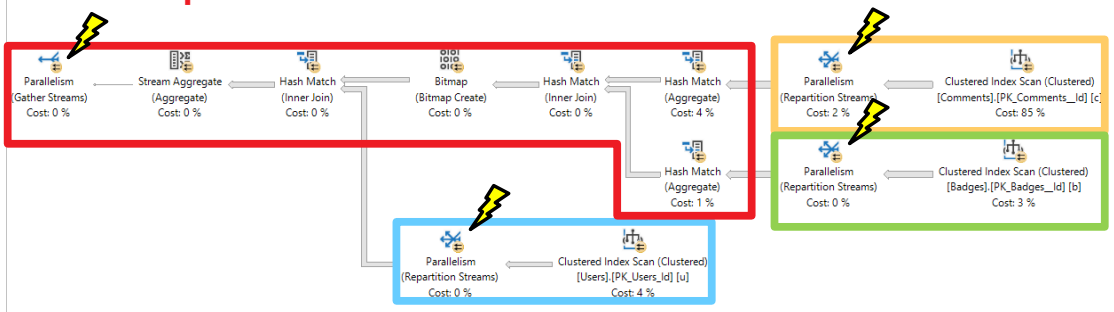
Queries can spawn more threads.

MAXDOP only controls how many cores a query uses.

It doesn't control:

- How many branches & threads get spawned
- The number of threads per parallel operator

Some parallel branches can run simultaneously.



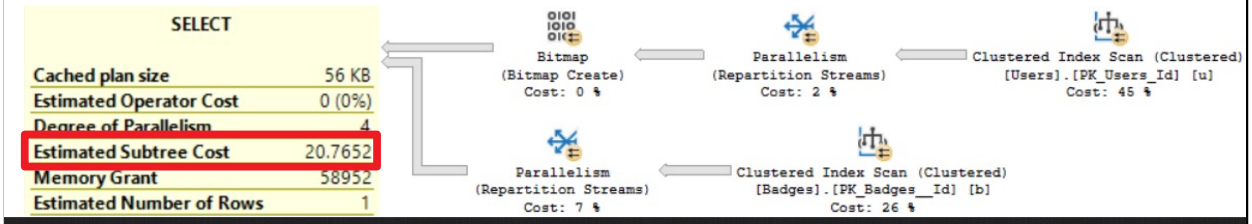
ThreadStat	
Branches	2
⊞ ThreadReservation	
UsedThreads	8

2.1 p20

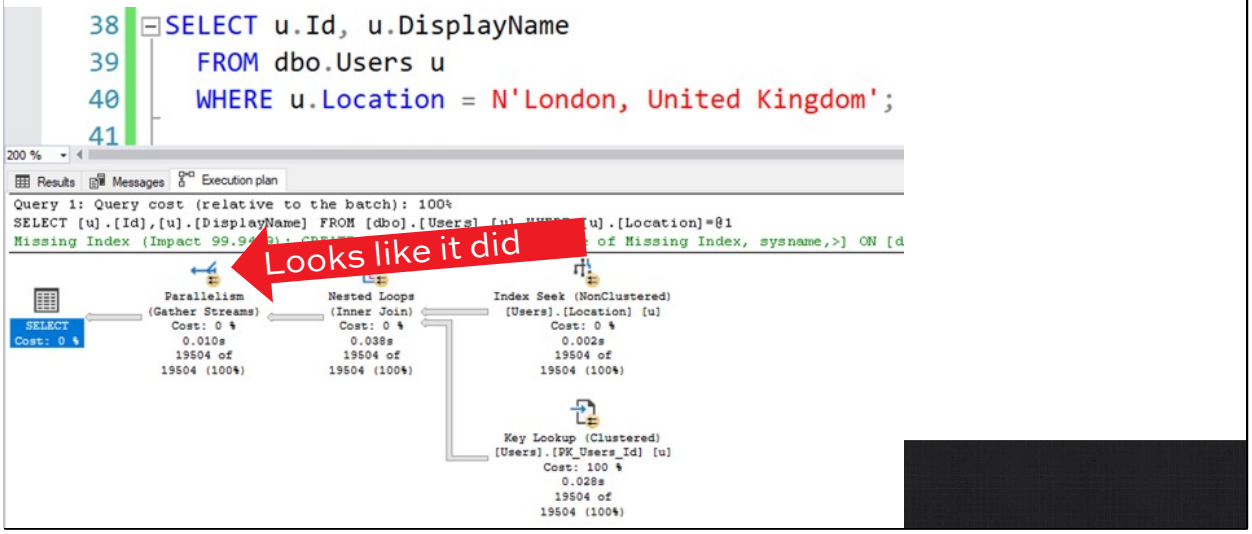
The parallel plan's cost might be < CTFP

Serial plan cost is what needs to exceed Cost Threshold

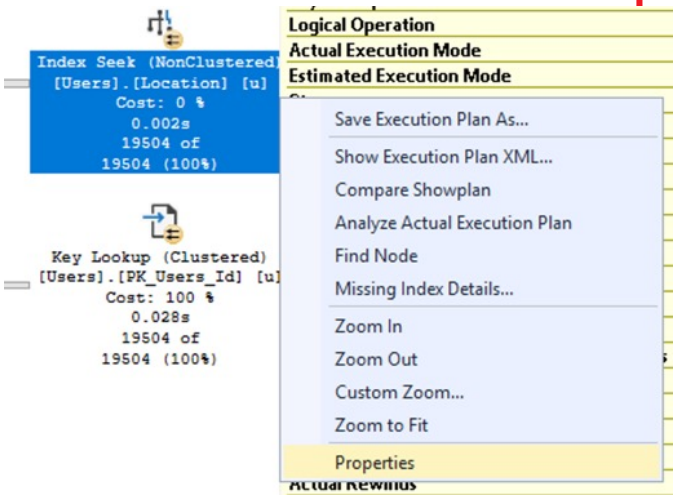
To see what the serial plan's cost was,
run it with an OPTION (MAXDOP 1) hint

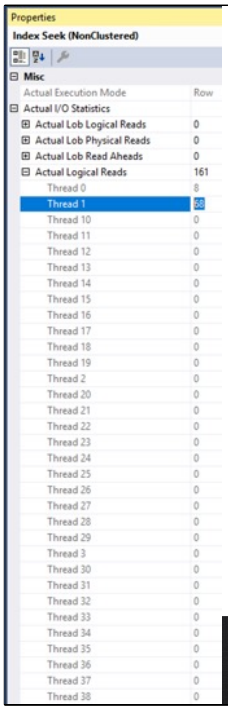


Back to our query. It went parallel, right?



Check the index seek properties...





Not really parallel.

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

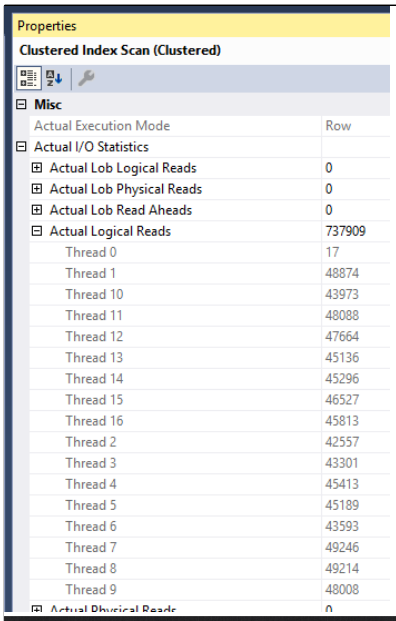
Parallelism is misunderstood.

It's NOT about
evenly dividing work across CPU cores.

It's about
making queries faster than if they ran on only 1 CPU core (and even that can go wrong.)

Why queries don't go evenly parallel

2.1 p26



Problem 1: dividing input

Sometimes pages aren't evenly divided.

The threads aren't doing the same amount of work.

They're not sharing CPU time or memory either: each one is pinned to a specific core, and given a specific amount of memory.

One thread can spill to disk while others don't.

One thread can compete with other tasks running on the same physical CPU core.

⊞ Actual Number of Batches	0
⊟ Actual Number of Rows	2
Thread 0	0
Thread 1	0
Thread 10	0
Thread 11	0
Thread 12	0
Thread 13	0
Thread 14	0
Thread 15	0
Thread 16	0
Thread 2	0
Thread 3	1
Thread 4	1
Thread 5	0
Thread 6	0
Thread 7	0
Thread 8	0
Thread 9	0
⊞ Actual Rebinds	0

Problem 2: uneven output

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

For example, if we don't have an index and we 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.

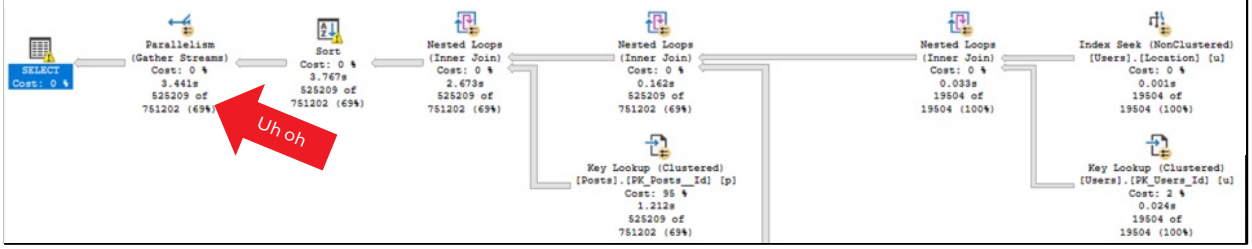
One thread might just find all of them.

When unpredictable output is bad

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

But what if *many* 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.



Problem 3: parallelism bugs

Index spools say they're parallel – but aren't.

Order preserving exchanges like parallel merge joins perform poorly.

We cover fixing these problems in MQT.
There's *nothing* you can do at the server level.

All we can do is limit their blast radius.

Measuring parallelism

2.1 p31

Wait stats reminder

What's Running Now

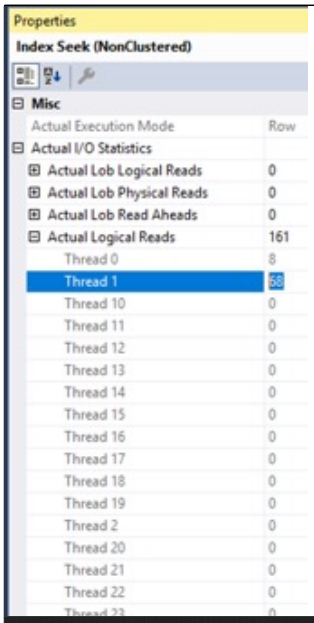
If a task is running, that's fine.

What's Waiting (Queue)

If a task is waiting on something like storage or CPU or locks, that's also fine.

2.1 p32

Assume we have the world's smallest SQL Server – it's only got one core.



All threads get tracked.

Thread 0: coordinating thread

- Works a little at the beginning to hand out work to other threads
- Then waits on all threads to finish

Thread 1: actually did the work

Threads 2+: didn't get assigned any work but we still have to track what they did (nothing)

2.1 p33

Thread 0's waits: harmless

Say we have a query go parallel across 8 cores.

The work is divided exactly evenly across them all.

All 8 cores work for 1 minute straight.

The coordinating thread is still waiting the whole time.

There is nothing wrong with that wait: it just means parallel queries are occurring.

Thread 2+'s waits: uneven work

Threads 2+ are said to be waiting, but they're not.

They can go on and do other stuff for other queries.

These waits can mean anything:

- Parallel work wasn't passed out evenly, or
- Threads found results unevenly, or
- There wasn't even enough work to get multiple threads involved (esp. parameter sniffing)

Tracking work

Microsoft finally fixed this in 2016 SP2 & 2017 CU3:

	Coordinating Thread	Idle Workers
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 p36

But they didn't really fix it.

CXCONSUMER is supposed to be totally ignorable...

But it's not. We've found queries that only produce CXCONSUMER during unbalanced parallelism.

CXPACKET is supposed to be bad...

But it's not, as long as we have plan reuse. A query compiled for big data will be unbalanced when it runs for tiny data.

The measurements aren't right.

When you see CXCONSUMER and CXPACKET waits, that doesn't mean you have a parallelism problem.

You might – but you might not.

So instead:

- Make sure you have good, sane settings
- Tune on queries doing a lot of work
- If you run across a leaderboard query with unbalanced parallelism, fix that per MQT

How to set CTFP & MAXDOP

2.1 p39

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.

2.1 p40

Setting Max Degree of Parallelism

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 p41

My workflow

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. (D.E.A.)
4. Let those settings bake in for a few days.
5. Look for queries still doing a lot of work, then tune them. (T.)
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 p42

After setting CTFP & MAXDOP

Let parallelism changes bake in for a few days, then:

D.E.A.: sp_BlitzIndex @GetAllDatabases = 1

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

Then let your new indexes bake in for a few days too.

After a few rounds of D.E.A.:

Do the T part of the D.E.A.T.H. Method:
tune indexes for specific queries by hand

Look at your top 10 leaderboard with sp_BlitzCache

When you run queries and get the actual plan, look for unbalanced parallelism as we discuss in MQT

There's no easy shortcut to fixing this

Server-wide CTFP & MAXDOP don't fix it

What's coming in SQL Server 2022

2.1 p45

Degree of Parallelism Feedback

New database-scoped config option:
DOP_FEEDBACK

- When a query has a parallel plan
- And it runs repeatedly
- And parallelism waits are a problem
- Each time it runs, DOP will decrease by 2, all the way down to 2

Where this makes sense

Microsoft's guidance in KB# 2806535 tells you to go all the way up to MAXDOP 16.

CPUs get more cores every year: there's a trend toward more cores, not faster cores.

SQL Server 2022 will be supported for a decade.

We might have 256-core CPUs – and MAXDOP might need to go higher than 16.

I’m hopeful, but…

DOP decisions are based on CX% measurements.

We’ve already covered how those are inaccurate.

It’s hard to make good decisions with bad data.

Recap

2.1 p49

Start with the basics.

Queries go parallel because they:

- Need to read a lot of 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. Tune that (D.E.A.T.H.), and parallelism drops
3. For the remaining queries (and those will be hard), review the abnormal parallelism causes & fixes from MQT