



GOTC 2023

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

OPEN SOURCE, INTO THE FUTURE

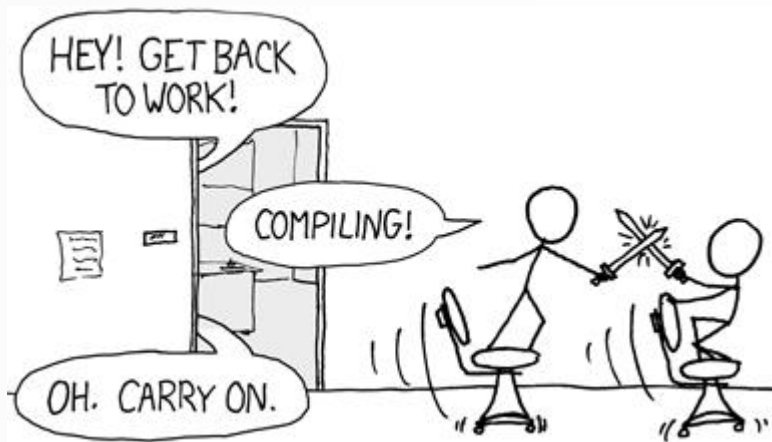
Rust并行编译的挑战与突破

李原 2022年5月28日

- 相关浅谈
- Rust并行编译的挑战与突破
- 从并行编译到并行程序设计
- Rust社区与并行编译

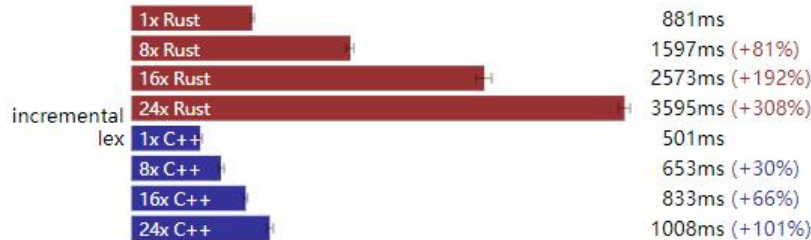
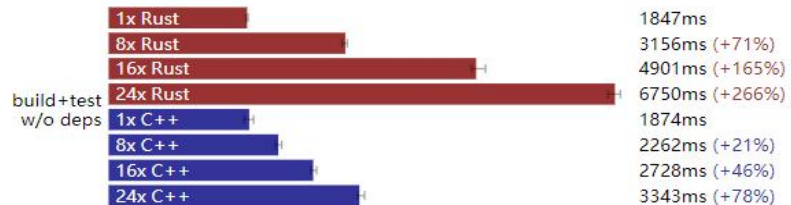
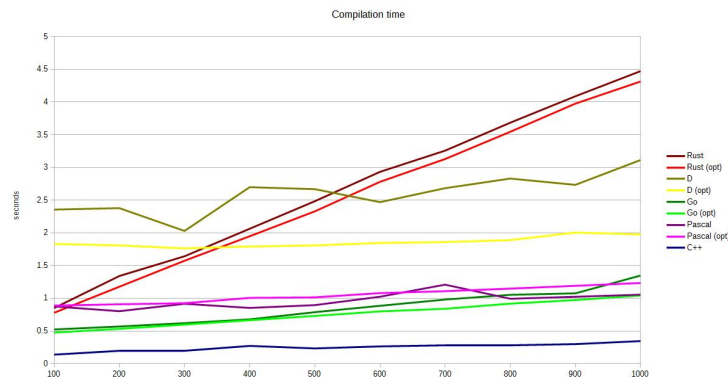
相关浅谈

编译器设计造成编译速度缓慢



- 单态化
- 借用检查
- 宏展开
- MIR优化
- ...

Rust规模编译速度慢于C++



提升编译效率成为近年社区重点工作

并行编译或成下一代编译效率突破利器

```

compare 2017-11-12 (e9f85430) 2019-07-24 (276a6304) 1 change
├─ inflate-opt      avg: -51.0%      min: -91.6%  max: -37.3%
├─ inflate-debug   avg: -47.1%      min: -91.5%  max: -28.8%
├─ tokio-webpush-simple-opt  avg: -34.9%      min: -81.7%  max: 15.0%
├─ crates.io-opt   avg: -53.0%      min: -80.0%  max: -17.5%
├─ futures-debug   avg: -42.1%      min: -78.1%  max: -6.7%
├─ style-servo-opt avg: -43.1%      min: -77.9%  max: -21.4%
├─ futures-opt     avg: -46.8%      min: -77.9%  max: -9.3%
├─ piston-image-opt  avg: -55.7%      min: -77.9%  max: -15.7%
├─ syn-debug       avg: -54.6%      min: -74.0%  max: -31.0%
├─ crates.io-debug avg: -50.9%      min: -73.8%  max: -25.2%
├─ syn-opt         avg: -37.8%      min: -73.7%  max: 14.0%
├─ piston-image-debug  avg: -45.1%      min: -73.5%  max: -24.9%
├─ regex-opt       avg: -30.4%      min: -73.2%  max: 24.4%
├─ deep-vector-opt  avg: -32.2%      min: -73.0%  max: -8.2%
├─ deep-vector-debug  avg: -42.0%      min: -72.9%  max: -32.4%
├─ encoding-debug  avg: -47.3%      min: -72.9%  max: -15.3%
├─ regression-1157-debug  avg: -41.3%      min: -72.8%  max: -46.3%
├─ tuple-stress-debug  avg: -37.2%      min: -72.7%  max: -25.2%
├─ script-servo-opt  avg: -23.9%      min: -72.7%  max: 35.7%
├─ encoding-opt    avg: -54.2%      min: -72.7%  max: -16.0%
├─ tuple-stress-opt  avg: -30.1%      min: -72.6%  max: -14.1%
├─ html5ever-debug  avg: -35.8%      min: -72.3%  max: -13.0%
├─ regression-1157-opt  avg: -27.2%      min: -72.0%  max: 49.1%
├─ html5ever-opt    avg: -44.3%      min: -71.6%  max: -17.2%
├─ regex-debug     avg: -56.6%      min: -71.6%  max: -18.0%
├─ hyper-debug     avg: -46.8%      min: -69.6%  max: -19.9%
├─ coercions-opt   avg: -25.0%      min: -67.9%  max: 3.2%
├─ hyper-opt       avg: -46.0%      min: -67.7%  max: -10.7%
├─ coercions-debug  avg: -28.8%      min: -65.3%  max: -4.8%
├─ unify-opt        avg: -57.0%      min: -62.2%  max: -50.6%
├─ unify-linearly-debug  avg: -58.0%      min: -61.8%  max: -51.4%
├─ helloworld-opt  avg: -54.1%      min: -58.9%  max: -47.3%
├─ helloworld-debug  avg: -53.2%      min: -58.6%  max: -47.4%
├─ tokio-webpush-simple-debug  avg: -45.1%      min: -45.1%  max: -25.4%
├─ unused-warnings-opt  avg: -19.9%      min: -33.9%  max: -4.9%
├─ unused-warnings-debug  avg: -21.6%      min: -32.5%  max: -8.0%

```

Compiler performance working group

Improving rustc compilation performance (build times)

Members

- Mark Rousskov**
GitHub: [Mark-Simulacrum](#)
Team leader
- Jakub Beránek**
GitHub: [Kobzol](#)
- Rémy Rakic**
GitHub: [lqd](#)
- Michael Woerister**
GitHub: [michaelwoerister](#)
- Nicholas Nethercote**
GitHub: [nnethercote](#)
- Felix Klock**
GitHub: [pnkfelix](#)
- Ryan Levick**
GitHub: [rlevy](#)
- Tyson Nottingham**
GitHub: [tgnottingham](#)
- Wesley Wiser**
GitHub: [wesleywiser](#)

Range	Mean	Count
[-0.5%, +25.23%]	+5.40%	217 (37)
[-59.83%, -0.47%]	-24.64%	233 (28)
[-59.83%, +25.23%]	-2.67%	450 (37)

Aggregations ▶

Newly broken benchmarks:

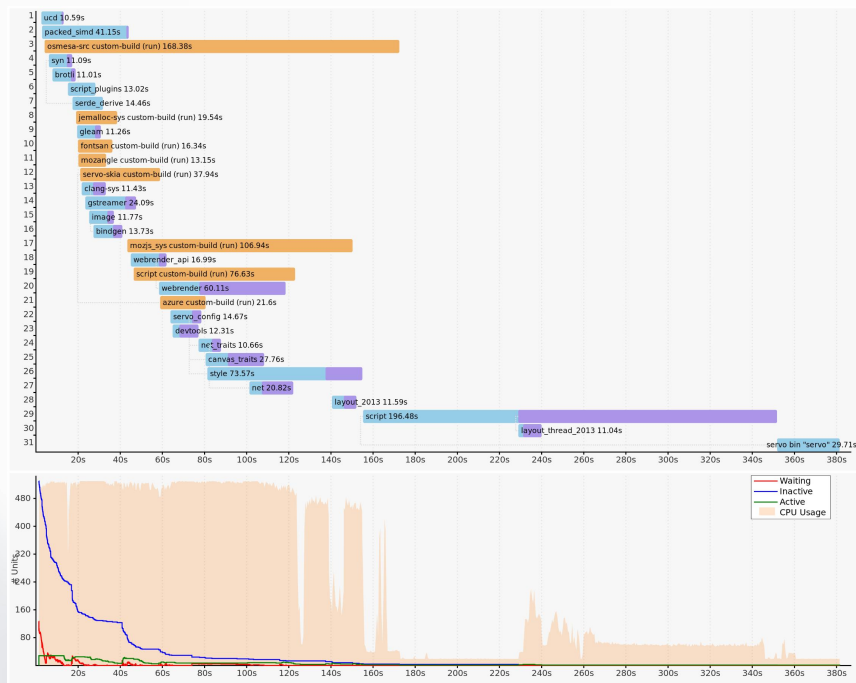
- html5ever
- tokio-webpush-simple
- webrender-vranch

Benchmark	Profile	Scenario	% Change	Significance Threshold	Significance Factor
webrender	check	full	-59.83%	0.20%	299.15x
webrender	check	incr-full	-59.67%	0.20%	298.36x
clap-rs	check	incr-full	-59.61%	0.20%	298.07x
piston-image	check	full	-59.15%	0.20%	295.72x
piston-image	check	incr-full	-59.14%	0.20%	295.72x
clap-rs	check	full	-58.58%	0.20%	292.91x
webrender-vranch	check	full	-56.84%	0.20%	284.22x
cargo	check	incr-full	-56.20%	0.20%	280.95x
cranelift-codegen	check	incr-full	-56.11%	0.20%	280.55x
cranelift-codegen	check	full	-56.06%	0.20%	280.28x
regex	check	incr-full	-56.01%	0.20%	280.06x
cargo	check	full	-55.77%	0.20%	278.85x
regex	check	full	-55.32%	0.20%	276.59x
webrender-vranch	check	incr-full	-54.92%	0.20%	274.58x
futures	check	full	-54.56%	0.20%	272.20x
futures	debug	incr-full	-54.46%	0.20%	272.28x
futures	debug	full	-54.19%	0.20%	270.97x
futures	check	incr-full	-53.88%	0.20%	269.40x
style-servo	check	full	-53.22%	0.20%	266.12x
serde	debug	full	-53.22%	0.20%	266.09x
clap-rs	debug	incr-full	-53.12%	0.20%	265.59x
style-servo	check	incr-full	-52.69%	0.20%	263.46x
cranelift-codegen	debug	full	-52.53%	0.20%	262.66x
regex	check	incr-full	-51.73%	0.20%	258.43x
hyper-2	debug	full	-51.66%	0.20%	258.32x
style-servo	debug	full	-51.57%	0.20%	258.66x
futures	opt	incr-full	-50.35%	0.20%	251.75x
hyper-2	check	full	-50.22%	0.20%	251.12x
hyper-2	check	incr-full	-49.99%	0.20%	249.97x
serde	opt	full	-49.81%	0.20%	249.03x
regex	check	full	-49.35%	0.20%	246.72x
syn	check	incr-full	-49.07%	0.20%	245.35x
futures	opt	full	-48.94%	0.20%	244.69x

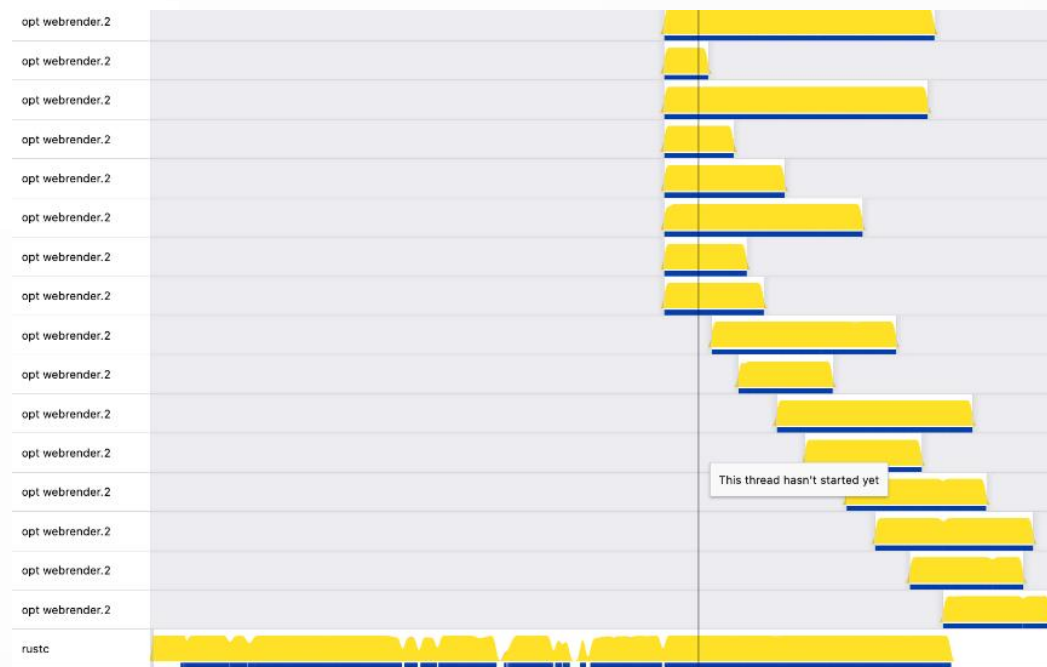
2017-2021, Rust编译速度已提升一倍以上

Rust社区编译器性能工作组

Cargo多crate并行

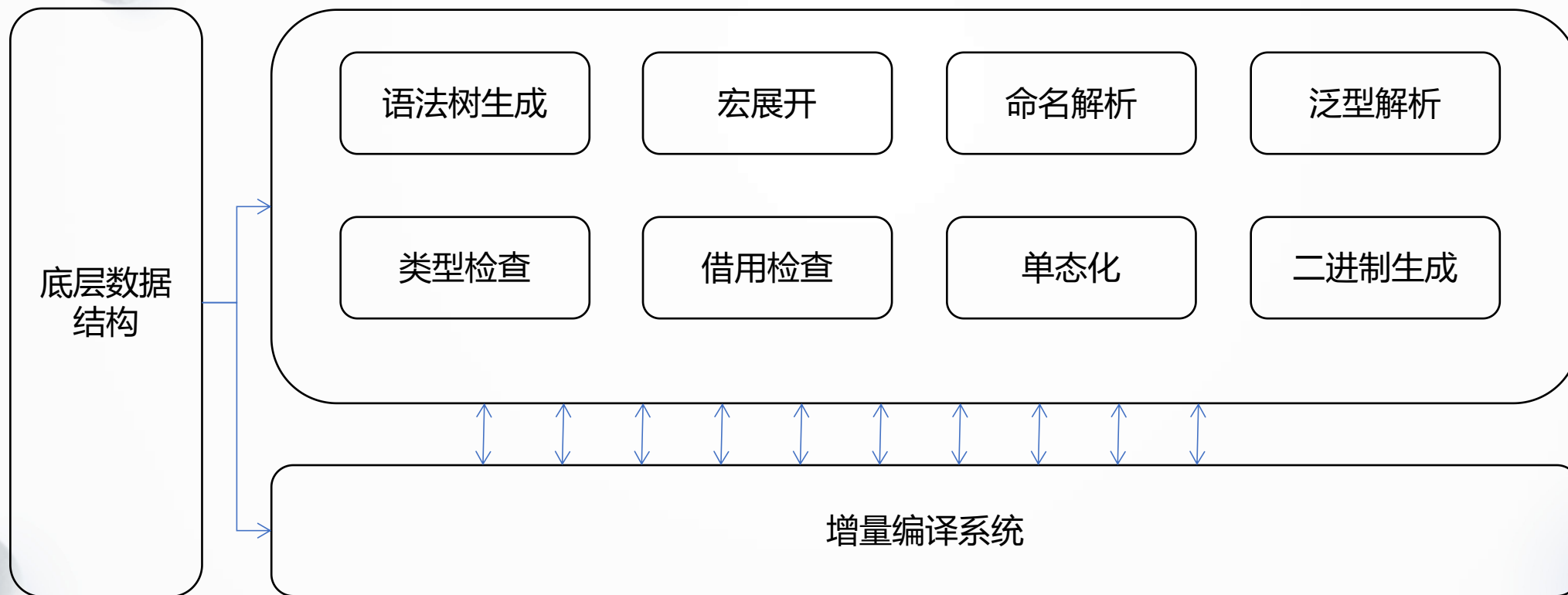


二进制生成并行



更多更好的并行化?

Rust语言编译器结构总览



考虑内部编译流程并行化

Rust并行并发

编译时线程安全检查

```
impl<T: ?Sized + Send> Send for RwLock<T>
impl<T: ?Sized + Send + Sync> Sync for RwLock<T>

impl<T: ?Sized + Send> Send for Mutex<T>
impl<T: ?Sized + Send> Sync for Mutex<T>

impl<T: Send> Send for Sender<T>
impl<T> !Sync for Sender<T>

impl<T: ?Sized + Sync + Send> Send for Arc<T>
impl<T: ?Sized + Sync + Send> Sync for Arc<T>
```

一些常见线程安全数据结构

```
use rayon::prelude::*;
fn sum_of_squares(input: &[i32]) -> i32 {
    input.par_iter() // <-- just change that!
        .map(|&i| i * i)
        .sum()
}
```



rayon-rs

```
#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let listener = TcpListener::bind("127.0.0.1:8080").await?;

    loop {
        let (mut socket, _) = listener.accept().await?;

        tokio::spawn(async move {
            let mut buf = [0; 1024];

            // In a loop, read data from the socket and write the data
            loop {
                let n = match socket.read(&mut buf).await {
                    // socket closed
                }
            }
        });
    }
}
```



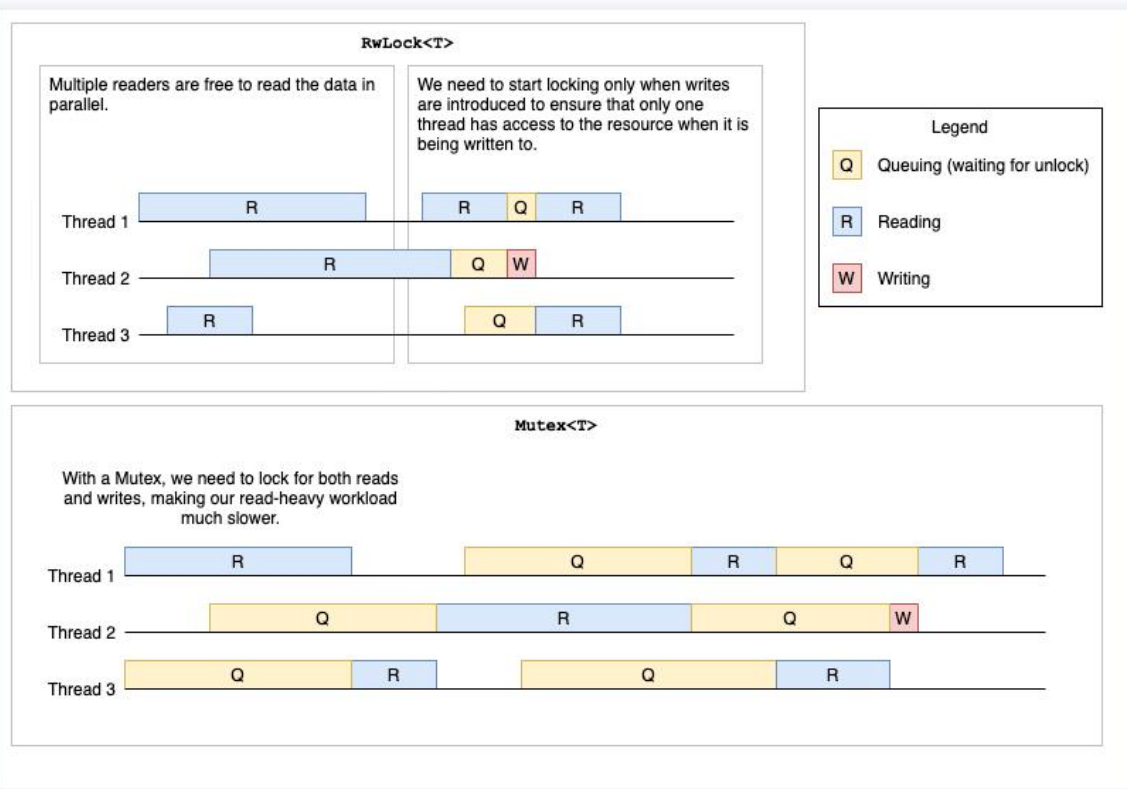
Tokio

Rust's asynchronous runtime.

常用Rust并行并发库

线程安全数据结构造成效率损失

增加程序复杂度



Mutex与RwLock

- 代码复杂度及健壮性
- benchmark资源限制
- profiling成本
- ...

Query/Function	Time (%)	Time (s)	Time delta	Executions
Totals	97.27%*	0.165	0.005 (3.4%)	52693
setup_global_ctxt	0.77%	0.001	0.001 (883.3%)	1
generate_crate_metadata	16.77%	0.028	0.001 (3.7%)	1
hir_crate	7.19%	0.012	0.001 (6.3%)	1
promoted_mir	4.19%	0.007	0.001 (9.6%)	0
metadata_decode_entry_optimized_mir	1.11%	0.002	0.000 (26.5%)	52
encode_query_results_for	6.67%	0.011	0.000 (3.1%)	55
expand_crate	15.31%	0.025	0.000 (1.3%)	1
incr_comp_query_cache_promotion	3.16%	0.005	0.000 (4.9%)	1
eval_to_const_value_raw	1.02%	0.002	0.000 (10.2%)	0
self_profile_alloc_query_strings	0.36%	0.001	0.000 (35.2%)	1
<unknown>	1.51%	0.002	0.000 (6.4%)	0
mir_for_ctfe	0.80%	0.001	0.000 (11.4%)	0
monomorphization_collector_graph_walk	1.31%	0.002	0.000 (5.3%)	1
incr_comp_load_dep_graph	1.34%	0.002	0.000 (5.1%)	1
blocked_on_dep_graph_loading	1.41%	0.002	0.000 (4.3%)	1
optimized_mir	0.94%	0.002	0.000 (4.8%)	52
crate_lints	0.85%	0.001	0.000 (5.2%)	1
type of	0.20%	0.000	0.000 (21.4%)	1405

rustc profiling 数据

Rust并行编译的挑战与突破

挑战：消减共享数据结构效率损失

	mean ^[1]	range	count ^[2]
Regressions ❌ (primary)	4.6%	[0.4%, 10.2%]	230
Regressions ❌ (secondary)	4.6%	[0.4%, 18.1%]	241
Improvements ✅ (primary)	-	-	0
Improvements ✅ (secondary)	-	-	0
All ❌✅ (primary)	4.6%	[0.4%, 10.2%]	230

Query/Function↓	Time (s)↓	Time (s)↓	Time delta▼
Totals	125.27s*	0.776	0.068 (9.6%)
incr_comp_intern_dep_graph_node	8.09%	0.063	0.013 (27.1%)
specialization_graph_of	5.14%	0.040	0.012 (43.7%)
check_mod_item_types	10.57%	0.082	0.007 (10.0%)
evaluate_obligation	9.77%	0.076	0.007 (10.0%)
typeck	5.49%	0.043	0.003 (7.0%)
generate_crate_metadata	2.51%	0.019	0.002 (11.9%)
mir_drops_elaborated_and_const_checked	3.13%	0.024	0.002 (8.1%)
mir_borrowck	5.05%	0.039	0.002 (4.7%)
mir_built	2.82%	0.022	0.002 (8.5%)
check_well_formed	1.78%	0.014	0.001 (9.9%)
param_env	1.35%	0.010	0.001 (9.3%)
mir_borrowck_const_arg	1.51%	0.012	0.001 (6.2%)
check_mod_privacy	0.61%	0.005	0.001 (15.3%)
encode_query_results_for	1.11%	0.009	0.001 (7.7%)
expand_crate	2.82%	0.022	0.001 (2.6%)
metadata_decode_entry_impl_trait_ref	0.98%	0.008	0.001 (7.5%)
lint_mod	0.40%	0.003	0.001 (19.8%)
predicates_defined_on	0.69%	0.005	0.000 (10.1%)
live_symbols_and_ignored_derived_traits	0.47%	0.004	0.000 (15.5%)
generics_of	0.73%	0.006	0.000 (9.3%)
type_of	0.78%	0.006	0.000 (8.3%)
collect_mod_item_types	0.44%	0.003	0.000 (15.6%)
incr_comp_encode_dep_graph	4.12%	0.032	0.000 (1.4%)
mir_for_ctfe	1.33%	0.010	0.000 (4.1%)
qualify_predicates_of	0.68%	0.004	0.000 (9.6%)

data structure	parallel	non-parallel
Lrc	std::sync::Arc	std::rc::Rc
Weak	std::sync::Weak	std::rc::Weak
Atomic{Bool}/{Usize}/{U32}/{U64}	std::sync::atomic::Atomic{Bool}/{Usize}/{U32}/{U64}	(std::cell::Cell<bool/usize/u32/u64>)
OnceCell	std::sync::OnceLock	std::cell::OnceCell
Lock<T>	(parking_lot::Mutex<T>)	(std::cell::RefCell)
RwLock<T>	(parking_lot::RwLock<T>)	(std::cell::RefCell)
MtRef<'a, T>	&'a T	&'a mut T
MtLock<T>	(Lock<T>)	(T)
ReadGuard	parking_lot::RwLockReadGuard	std::cell::Ref
MappedReadGuard	parking_lot::MappedRwLockReadGuard	std::cell::Ref
WriteGuard	parking_lot::RwLockWriteGuard	std::cell::RefMut
MappedWriteGuard	parking_lot::MappedRwLockWriteGuard	std::cell::RefMut
LockGuard	parking_lot::MutexGuard	std::cell::RefMut
MappedLockGuard	parking_lot::MappedMutexGuard	std::cell::RefMut
MetadataRef	OwningRef<Box<dyn Erased + Send + Sync>, [u8]>	OwningRef<Box<dyn Erased>, [u8]>

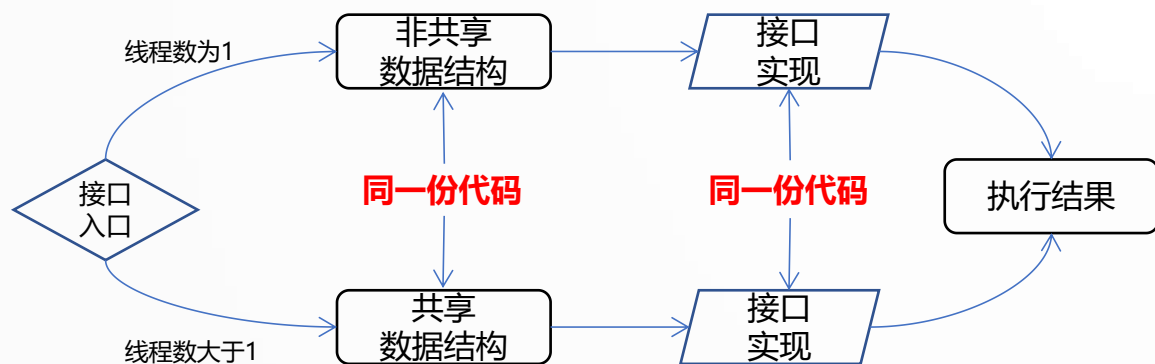
基于条件编译的共享数据结构实现

共享数据结构的性能损耗问题

缺点：用户需自行生成编译器

挑战：消减共享数据结构效率损失

Specailization —— 基于GAT的共享数据结构实现



```
pub trait SLock {
    type Lock<T>;

    type LockGuard<'a, T: 'a>: DerefMut<Target = T>;

    fn new<T>(value: T) -> Self::Lock<T>;

    fn lock<'a, T>(l: &'a Self::Lock<T>) -> Self::LockGuard<'a, T>;

    fn try_lock<'a, T>(l: &'a Self::Lock<T>) -> Option<Self::LockGuard<'a, T>>;

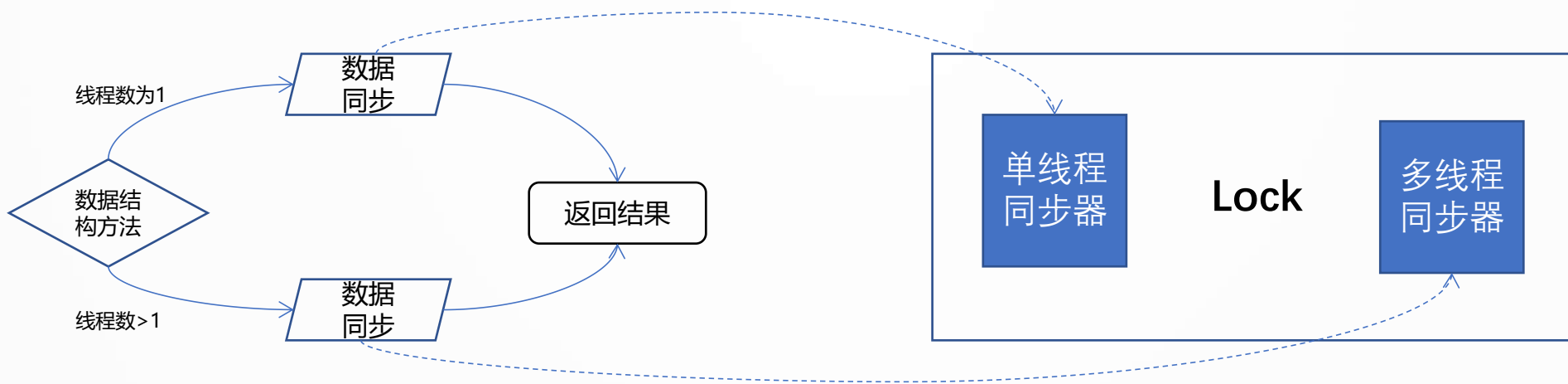
    fn lock_inner<T>(l: Self::Lock<T>) -> T;
}

struct DepGraphData<K: DepKind, L: SLock> {
    current: CurrentDepGraph<K, L>,
    dep_node_debug: L::Lock<FxHashMap<DepNode<K>, String>>,
    debug_loaded_from_disk: L::Lock<FxHashSet<DepNode<K>>>,
}
```

缺点：业务代码带有泛型参数

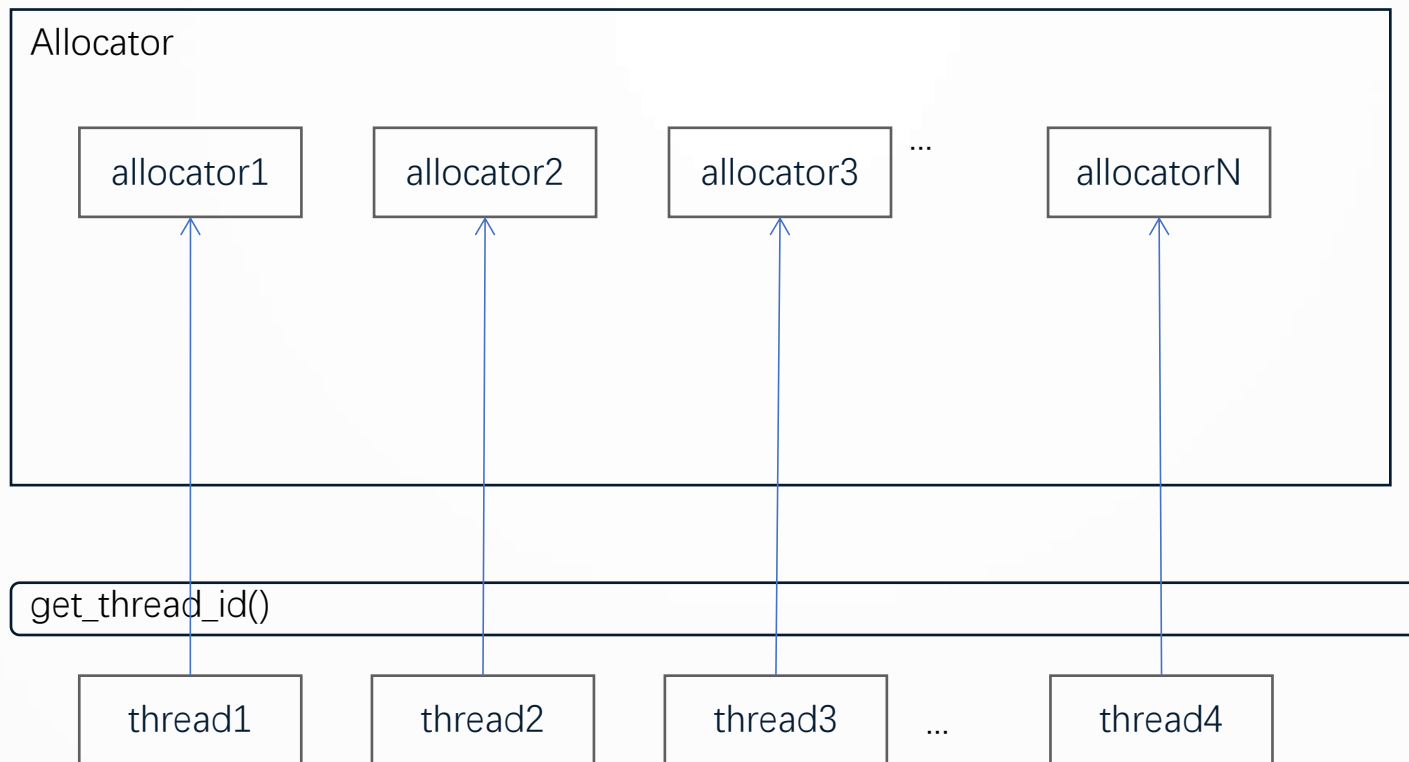
挑战：消减共享数据结构效率损失

动态线程安全检查 —— 自动切换数据同步模式



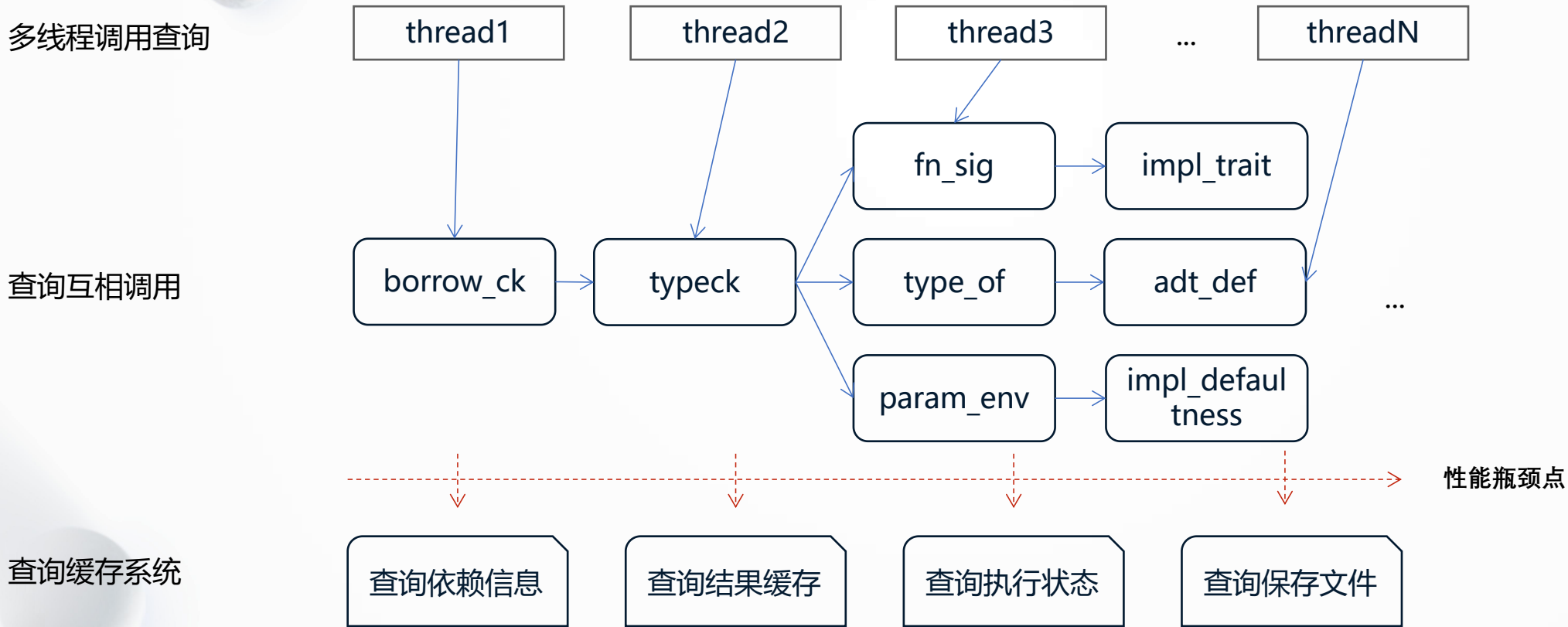
挑战：并行环境内存分配器的设计

多线程内存分配器，线程分而治之



WorkerLocal数据结构设计

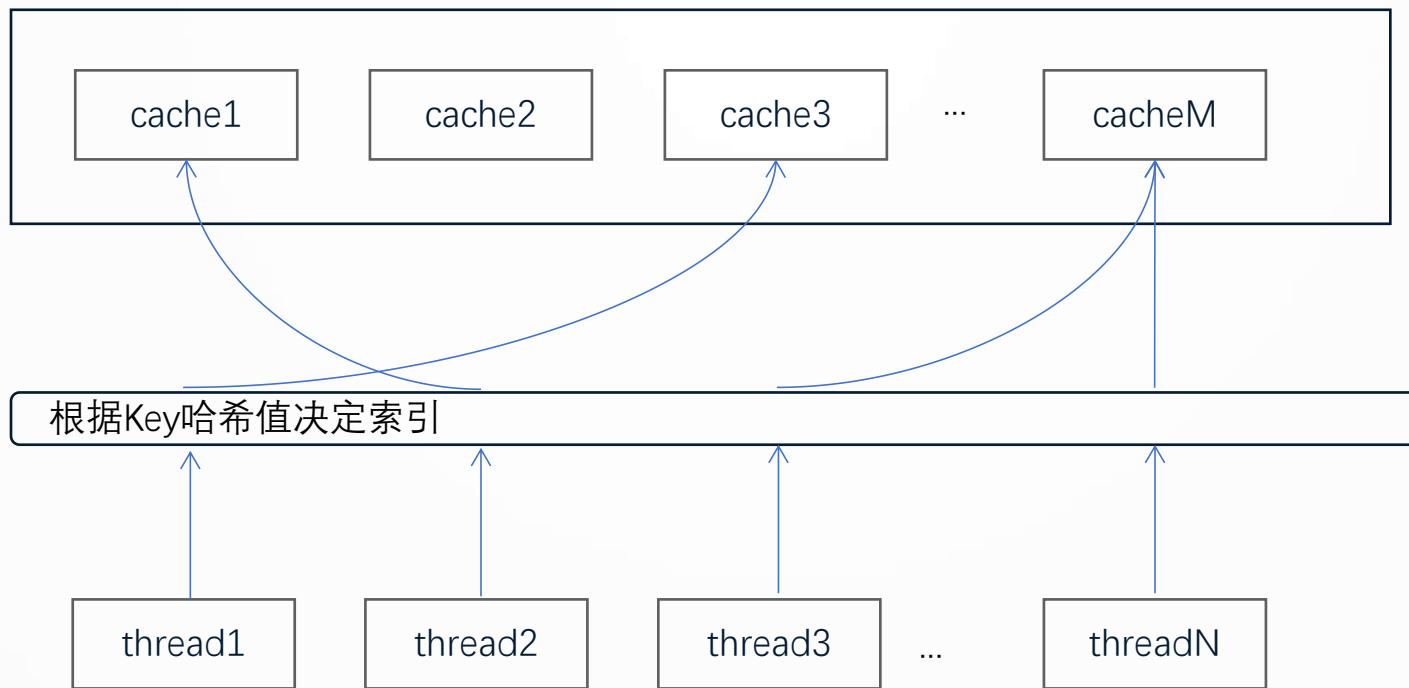
挑战：缓存系统访问热点效率瓶颈



频繁访问查询系统，成为效率瓶颈点

挑战：缓存系统访问热点效率瓶颈

Sharded —— 接口统一，存储分离



Sharded数据结构设计

- 抹平共享数据结构造成的性能差距
- 多线程环境下的编译器度量方案
- 针对并行环境的编译器测试
- 深化编译器并行化

从并行编译到并行程序设计

用巧妙的数据结构设计化解效率瓶颈

```
use std::cell::RefCell;
thread_local! {
    pub static F00: RefCell<u32> = RefCell::new(1);

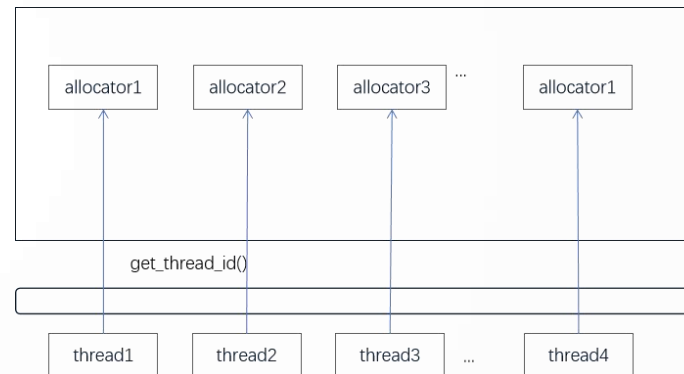
    #[allow(unused)]
    static BAR: RefCell<f32> = RefCell::new(1.0);
}
```

- thread_local

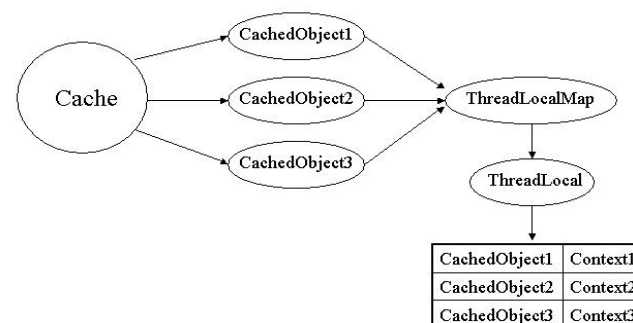
```
struct DepGraphData<K: DepKind, L: SLock> {
    current: CurrentDepGraph<K, L>,
    dep_node_debug: L::Lock<FxHashMap<DepNode<K>, String>>,
    debug_loaded_from_disk: L::Lock<FxHashSet<DepNode<K>>>,
}
pub struct DepGraph<K: DepKind> {
    data: Option<Lrc<DepGraphData<K, SRefCell>>>,
    shared_data: Option<Lrc<DepGraphData<K, SMutex>>>,

    /// This field is used for assigning DepNodeIndices when running in
    /// non-incremental mode. Even in non-incremental mode we make sure that
    /// each task has a "DepNodeIndex" that uniquely identifies it. This unique
    /// ID is used for self-profiling.
    virtual_dep_node_index: Lrc<AtomicU32>,
}
```

- specialization



- WorkerLocal



- 线程级缓存

用优秀的设计消减数据同步代价



```
#[derive(Debug)]
pub struct TransitiveRelation<T> {
    // Frozen transitive relation elements and edges.
    builder: Frozen<TransitiveRelationBuilder<T>>,

    // This is a cached transitive closure derived from the edges.
    // Currently, we build it lazily and just throw out any existing
    // copy whenever a new edge is added. (The Lock is to permit
    // the lazy computation.) This is kind of silly, except for the
    // fact its size is tied to 'self.elements.len()', so I wanted to
    // wait before building it up to avoid reallocating as new edges
    // are added with new elements. Perhaps better would be to ask the
    // user for a batch of edges to minimize this effect, but I
    // already wrote the code this way. :P -nmatsakis
    closure: Lock<Option<BitMatrix<usize, usize>>>,
}
```

```
#[derive(Clone, Debug)]
pub struct TransitiveRelationBuilder<T> {
    // List of elements. This is used to map from a T to a usize.
    elements: FxIndexSet<T>,

    // List of base edges in the graph. Require to compute transitive
    // closure.
    edges: FxHashSet<Edge>,
}

#[derive(Debug)]
pub struct TransitiveRelation<T> {
    // Frozen transitive relation elements and edges.
    builder: Frozen<TransitiveRelationBuilder<T>>,

    // Cached transitive closure derived from the edges.
    closure: Frozen<BitMatrix<usize, usize>>,
}
```

读写分离

```
#[derive(Default)]
pub struct HygieneEncodeContext {
    pub serialized_ctxts: Lock<FxHashSet<SyntaxContext>>,
    pub latest_ctxts: Lock<FxHashSet<SyntaxContext>>,
}
```

```
#[derive(Default)]
pub struct HygieneEncodeContext {
    pub serialized_ctxts: FxHashSet<SyntaxContext>,
    pub latest_ctxts: FxHashSet<SyntaxContext>,
}
```

```
impl<'a, 'tcx> Encodable<CacheEncoder<'a, 'tcx>> for SyntaxContext {
    fn encode(&self, s: &mut CacheEncoder<'a, 'tcx>) {
        if !s.hygiene_context.serialized_ctxts.lock().contains(&self) {
            s.hygiene_context.latest_ctxts.lock().insert(&self);
        }
        ctxt.0.encode(s);
    }
}
```

```
impl<'tcx> Encodable<CacheEncoder<'tcx>> for SyntaxContext {
    fn encode(&self, s: &mut CacheEncoder<'tcx>) {
        if !s.hygiene_context.serialized_ctxts.contains(&self) {
            s.hygiene_context.latest_ctxts.insert(&self);
        }
        self.0.encode(s);
    }
}
```

限制作用范围

```
/// Info about a parsing session.
pub struct ParseSess {
    pub gated_spans: GatedSpans,
    // ...
}
```

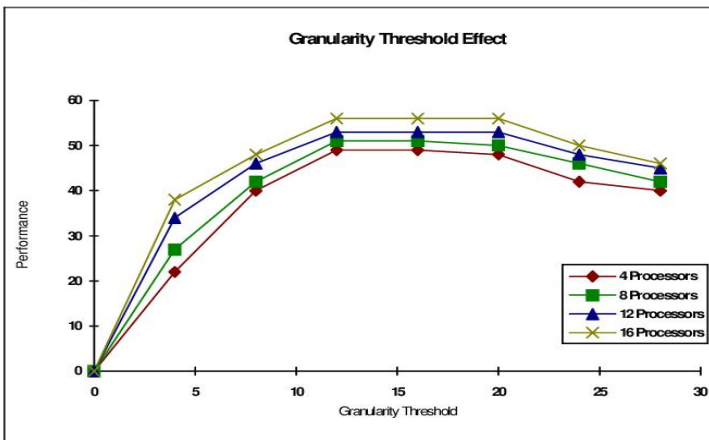
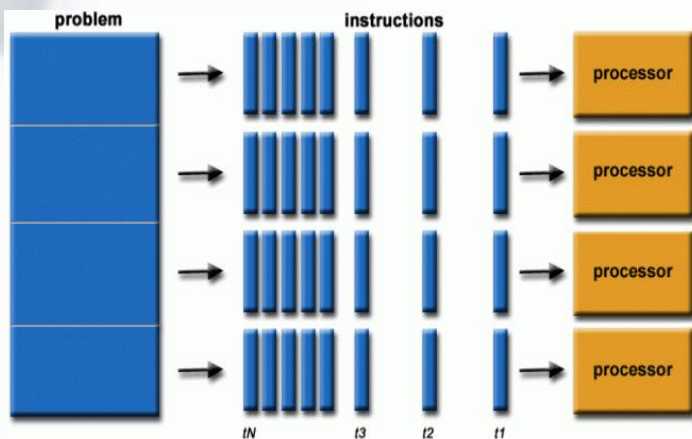
```
pub struct GatedSpans {
    pub spans: Lock<FxHashMap<Symbol, Vec<Span>>>,
}
```

```
#[derive(Clone)]
pub struct Parser<'a> {
    pub sess: &'a ParseSess,
    // ...

    pub gated_spans: Rc<RefCell<GatedSpans>>,
}
```

```
pub struct GatedSpans {
    pub spans: FxHashMap<Symbol, Vec<Span>>,
}
```

COPY-WRITE机制



任务粒度与并行效率的关系

```
fn weight_max(self) -> Weight<Self>
```

Shorthand for `self.weight(f64::INFINITY)`. This forces the smallest granularity of parallel execution, which makes sense when your parallel tasks are (potentially) very expensive to execute.

rayon库中的粒度控制函数

```
pub struct ThreadPoolBuilder<S = DefaultSpawn> {
    /// The number of threads in the rayon thread pool.
    /// If zero will use the RAYON_NUM_THREADS environment variable.
    /// If RAYON_NUM_THREADS is invalid or zero will use the default.
    num_threads: usize,

    /// Custom closure, if any, to handle a panic that we cannot propagate
    /// anywhere else.
    panic_handler: Option<Box<PanicHandler>>,

    /// Closure to compute the name of a thread.
    get_thread_name: Option<Box<dyn FnMut(usize) -> String>>,

    /// The stack size for the created worker threads
    stack_size: Option<usize>,

    /// Closure invoked on deadlock.
    deadlock_handler: Option<Box<DeadlockHandler>>,
}

pub fn mark_blocked(&self, deadlock_handler: &Option<Box<DeadlockHandler>>) {
    let mut data : MutexGuard<SleepData> = self.data.lock().unwrap();
    debug_assert!(data.active_threads > 0);
    debug_assert!(data.blocked_threads < data.worker_count);
    debug_assert!(data.active_threads > 0);
    data.active_threads -= 1;
    data.blocked_threads += 1;

    data.deadlock_check(deadlock_handler);
}
```

基于rayon线程池的自动触发式死锁检测

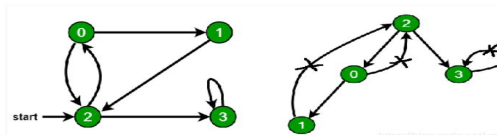
```
fn cycle_check(
    query_map: &QueryMap,
    query: QueryJobId,
    span: Span,
    stack: &mut Vec<(Span, QueryJobId)>,
    visited: &mut FxHashSet<QueryJobId>,
) -> Option<Option<Waiter>> {
    if !visited.insert(query) {
        return if let Some(p) = stack.iter().position(|q| q.1 == query) {
            // We detected a query cycle, fix up the initial span and return Some
            // Remove previous stack entries
            stack.drain(..p);
            // Replace the span for the first query with the cycle cause
            stack[0].0 = span;
            Some(visited.get(&query).unwrap())
        } else {
            None
        };
    }

    // Query marked as visited is added it to the stack
    stack.push((span, query));

    // Visit all the waiters
    let r = visit_waiters(query_map, query, |span: Span, successor: QueryJobId| {
        cycle_check(query_map, query: successor, span, stack, visited)
    });

    // Remove the entry in our stack if we didn't find a cycle
    if r.is_none() {
        stack.pop();
    }

    r
}
```



通过有向图环路检测移除死锁

Rust社区与并行编译

Rust社区与并行编译



Parallellizing rustc using Rayon
compiler

Zoxc Jan '18

Parallellizing rustc using Rayon

My plan for parallellizing rustc basically is to allow use of Rayon anywhere in the compiler. If you're unfamiliar with Rayon or need a refresher, I suggest you read <http://smallcultfollowing.com/babysteps/blog/2015/12/18/rayon-data-parallelism-in-rust/> 317.

There are a number of loops in rustc which just iterates over all items in a crate and these should use Rayon iterators instead. Parallellizing the loops for type checking and borrow checking are particularly impactful. This proposal does not alter the structure of the compiler which means it's reasonably easy to implement (unlike MIR or incremental compilation). You can find an implementation of most of this proposal at <https://github.com/Zoxc/rust/tree/rayon-queries> and <https://github.com/Zoxc/rayon/tree/riber> for the Rayon changes.

18年1月由编译器团队成员提出

rustc 并行化工作组

让 rustc 默认支持并行编译

RUSTC 并行化工作组 成员

ZULIP 上的 #T-COMPILER / WG-PARALLEL-RUSTC

成员

- Mark Rousskov**
GitHub: Mark-Simulacrum
团队领导
- Alex Crichton**
GitHub: alexcrichton
- Santiago Pastorino**
GitHub: spastorino
- Niko Matsakis**
GitHub: nikomatsakis
团队领导
- Josh Stone**
GitHub: cuxiper
- Zoxc**
GitHub: Zoxc

成立社区工作组

Query Parallelization Tracking Issue #48685

Open 20 of 31 tasks nikomatsakis opened this issue on Mar 3, 2018 · 10 comments

nikomatsakis commented on Mar 3, 2018 · edited by TaKO8Ki

This issue is a sub-issue of #48547: it tracks the in-progress effort to parallelize rustc across queries. This work is being spearheaded by @Zoxc.

Goals

Allow rustc to execute queries in parallel with one another. Enable the use of rayon or other tools for intra-query parallelization as well. See this [internals thread](#) for more information.

Release nightly compilers with ability to internally parallelize #591

Open alexcrichton opened this issue on Apr 3, 2019 · 3 comments

alexcrichton commented on Apr 3, 2019 · edited

This is intended to be a tracking issue to releasing nightly compilers with the ability to internally parallelize themselves but they are still defaulted to single threaded mode. This is part of the larger [parallel compiler tracking issue](#), and is intended to be an incremental step towards fully closing that out.

A recent attempt to build binaries of the parallel compiler led to the thought of whether we could just enable a parallel compiler by default. Note that there are two axes we can change here over time:

- Whether or not the compiler can be parallelized at all, aka whether it's built with the `--cfg parallel_compiler` flag.
- Whether or not the compiler by default is parallelized, aka the default value of `-Z threads`.

The proposal in this issue is to default to `-Z threads=1` (or the moral equivalent) but build nightly compilers with `--cfg parallel_compiler` (or the equivalent thereof). The intention is to get us closer to shipping a parallel compiler while buying us time to continue to fix any issues that arise. This would allow, for example, for users to very easily test out parallel compilation locally by using `RUSTFLAGS=-Zthreads=16`.

The main blocker for doing this is performance. Requested in a [recent thread](#) we realized it's imported to not watch the [comparison of instruction counts](#) but rather instead watch the [wall time numbers](#). The instruction count numbers regress 2-3% which looks deceptively good, but the wall-time numbers regress 10-20% (ish) which is much more serious.

Some further investigation shows that most of the slowdown is likely coming from the use of [mutexes](#) (as opposed to other avenues like removing parallel code, the overhead of using `rayon`, or using `Arc` instead of `Rc`).

The next steps here would be to investigate whether we can recover the performance lost from using mutexes (probably if we can remove the mutexes one way or another).

This issue will likely receive many updates over time!

陷入技术阻塞，人员逐渐流失

Reboot Parallel Rustc WG Proposal #567

Closed 3 tasks done SparrowLii opened this issue 15 days ago · 5 comments

SparrowLii commented 15 days ago · edited by nethercote

Proposal

Reboot the parallel compilation working group, make the current parallel compiler truly available to Rust users, and improve the functions and supporting facilities of the parallel compiler.

Motivation

Parallelization is an important way to improve the efficiency of the compiler. The parallel rustc working group in the compiler team is responsible for the parallelization of the compiler. At current the working group has been stagnant for a long time due to the exit of the main developer.

Yet the parallel compiler still has considerable value and potential. In the [previous build test](#), quite a few parts of the test set achieved huge improvements in wall-time. And many parts of the compilation process, e.g. expansion, have the potential to be parallelized.

Design

解决技术阻塞，工作组重启

Rust Compiler Ambitions for 2022

Feb. 22, 2022 · Felix Klock, Wesley Wiser on behalf of The Compiler Team

Rust Compiler Ambitions for 2022 Parallel Compilation

Parallel Compilation is one avenue for improving compiler performance. It is also a very complex area, especially when it comes to the tradeoff of how much of a hit one is willing to take on single core builds in order to enable more parallel computation. We already parallelize our LLVM invocations, but the parallelization of the rest of the compiler remains in an experimental state. This is an area we think needs long-term collaborative effort with the compiler team. We do not expect to deliver a solution here this year.

If you want to discuss more with us about past attempts and ideas for the future, please reach out to [pnkfelix](#) and [wesleywiser](#).

编译器Roadmap寻求帮助

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

THANKS