devi v0.4.0 - Comprehensive Performance Characterization

This release contains three independent performance studies on parallel chess search:

1. Heterogeneous Core Scheduling

- **Problem**: Can we exploit Apple Silicon's P/E asymmetry for chess search?
- **Method**: QoS-based thread biasing across 4 policies
- **Result**: **E-cores 12.8x slower**, mixed policy achieves only 65% of expected
- **Details**: core_pinning_analysis.md

2. Scaling Laws (Amdahl vs Gustafson)

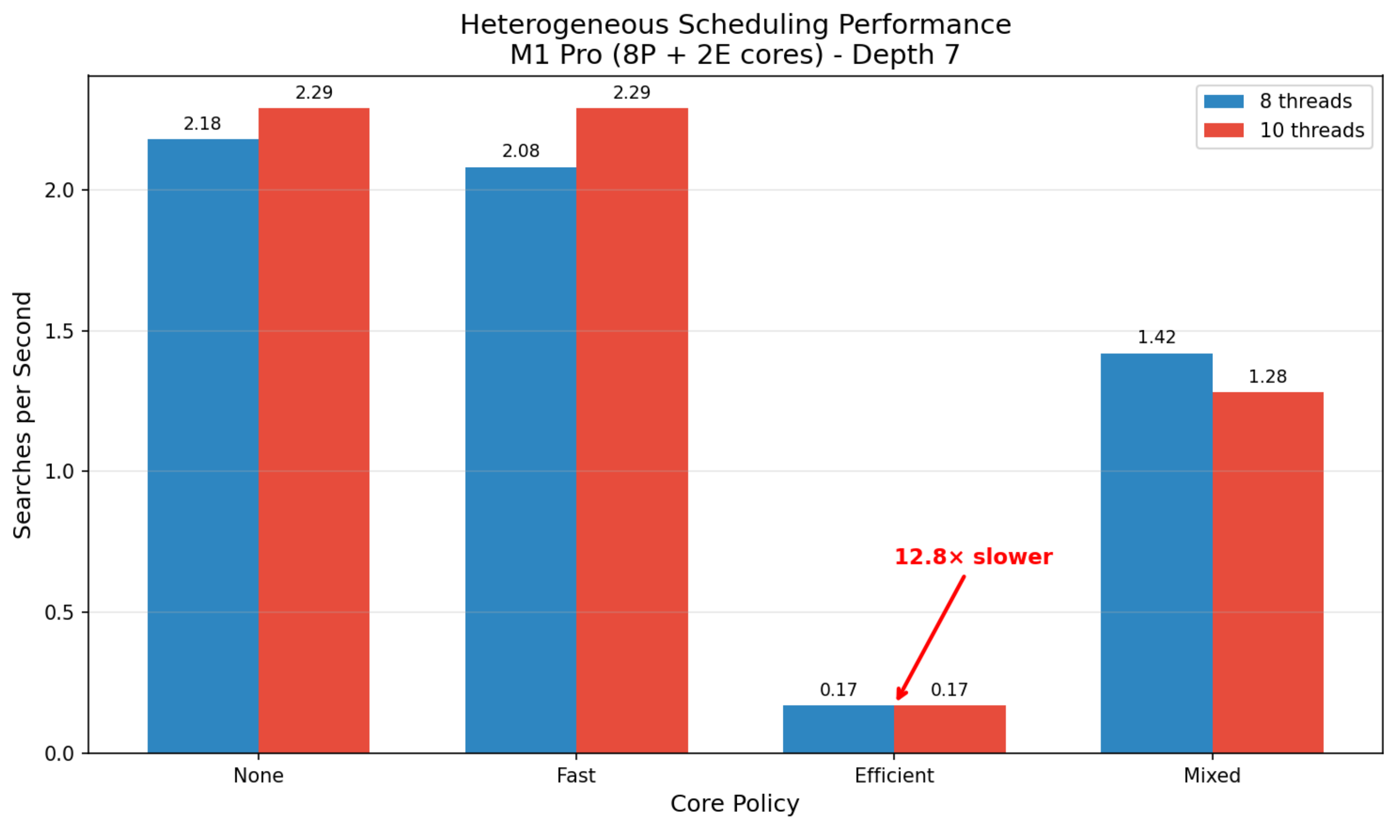
- **Problem**: Does chess search follow Amdahl's Law (fixed problem) or Gustafson's Law (scaled problem)?
- **Method**: Compare scaling at depth 4 vs depth 7
- **Result**: **48% serial fraction reduction** (0.27 -> 0.14) as depth increases
- **Details**: scaling_analysis.md

3. Fault Tolerance Baseline

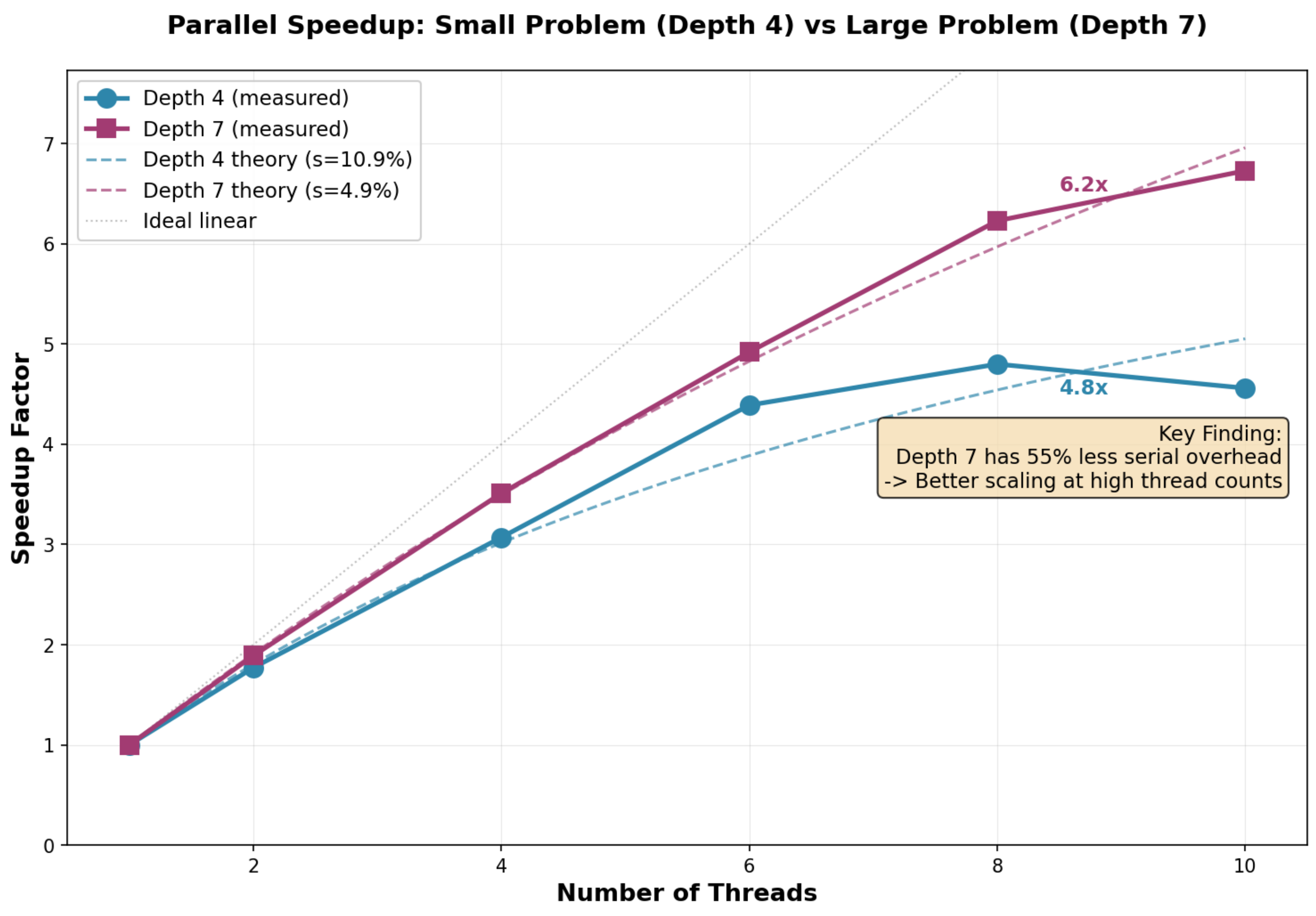
- **Problem**: What's the cost of thread-level fault recovery without checkpointing?
- **Method**: Inject panic at move 5 after 2-ply work, measure full retry overhead
- **Result**: **100% overhead** - Rayon's all-or-nothing model discards all parallel work
- **Details**: fault_tolerance_analysis.md

Summary

Heterogeneous Performance



Scaling Comparison



Key Findings

1. **Fault recovery needs checkpointing** - 100% overhead unacceptable for production
2. **Larger problems scale better** - Gustafson's Law dominates at higher depths
3. **E-cores unsuitable for branch-heavy work** - 12.8x gap motivates separate work queues

- Apple M1 Pro (8P + 2E cores)

Reproduce All Studies

```
 # Clone and build
git clone https://github.com/Sid4mn/devi-chess-engine.git
cd chess-engine-rust && cargo build --release

# 1. Fault tolerance
./target/release/devi --fault-analysis --depth 7 --threads 8

# 2. Scaling laws
./scripts/analysis/multi_depth_scaling.sh

# 3. Heterogeneous scheduling
./scripts/heterogeneous.sh
```

Data Files

- Fault tolerance: `benchmarks/fault_overhead.csv`
- Heterogeneous: `benchmarks/hetero_*.csv`
- Scaling: `benchmarks/scaling_analysis/`

Future Work: Harness-Inspired Lightweight Scheduler

1. Our two-phase root execution design adapts concepts from Belviranli et al.'s Harness framework [1], which demonstrated that **pattern-based work classification** can effectively utilize heterogeneous cores. While Harness targets MapReduce workloads with runtime pattern detection, we apply similar principles to chess search:

- **Harness approach**: Detect map/reduce patterns -> route to appropriate cores
- **Our adaptation**: PV probe estimates subtree complexity -> route heavy/light moves to P/E pools

The key insight from Harness is that **workload characterization enables better heterogeneous scheduling** which directly motivates our PV probe design.

[1] Belviranli, M. E., et al. "Harness: A Pattern-based Framework for Optimizing Heterogeneous MapReduce Applications." ICS 2025.

2. **Checkpoint Recovery** - Per-thread root-level checkpointing with `catch_unwind` to preserve completed move evaluations on panic. Target: ≤30% overhead (vs current 100% full retry).