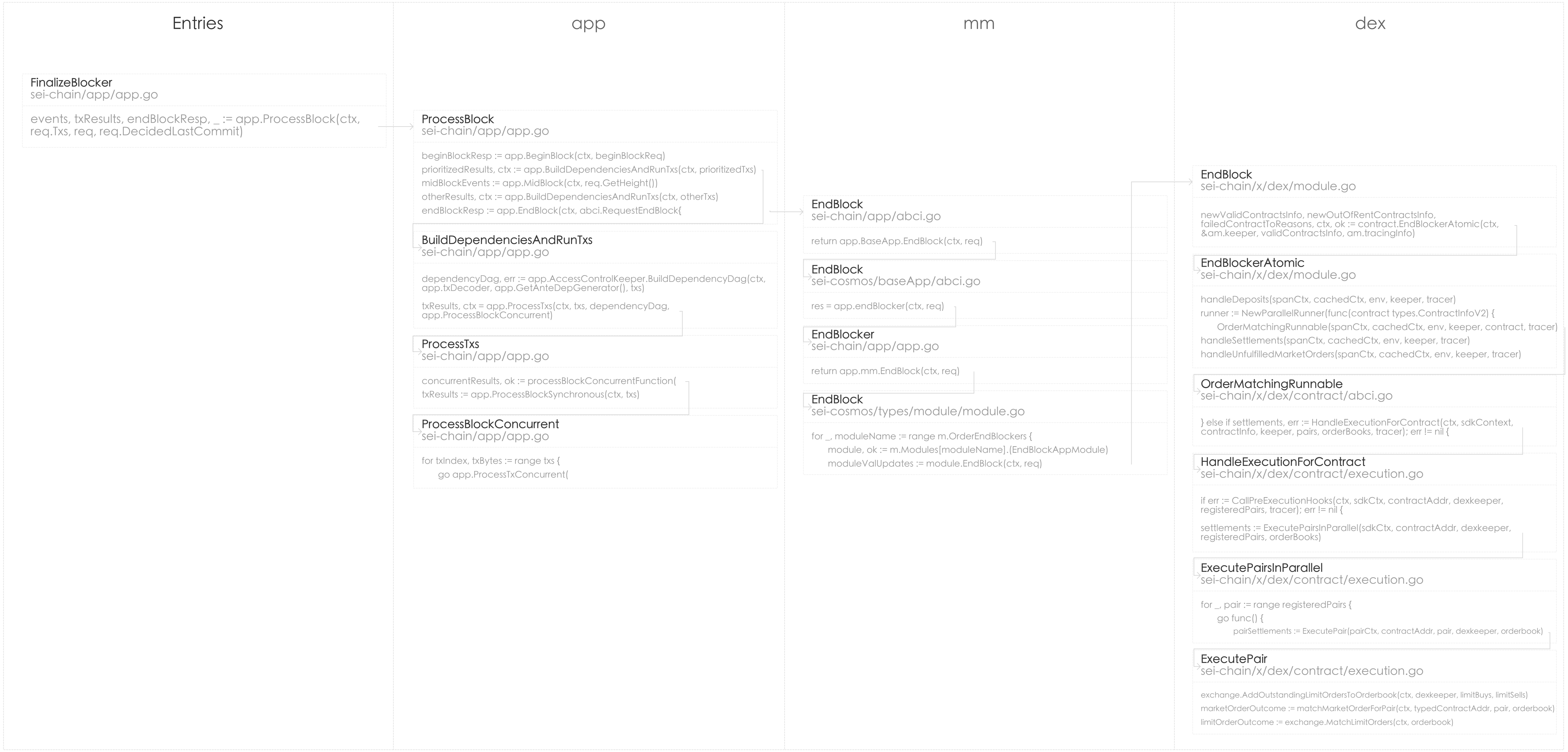
sei No.1 txs and orderbook parallelization

https://github.com/xiaoyu1998

Resolution:2560×1440(2K)
https://github.com/sei-protocol/sei-chain/tree/v3.5.0



Entries

FinalizeBlocker
sei-chain/app/app.go

```
events, txResults, endBlockResp, _ := app.ProcessBlock(ctx,
req.Txs, req, req.DecidedLastCommit)
```

app

ProcessBlock
sei-chain/app/app.go

```
beginBlockResp := app.BeginBlock(ctx, beginBlockReq)
prioritizedResults, ctx := app.BuildDependenciesAndRunTxs(ctx, prioritizedTxs)
midBlockEvents := app.MidBlock(ctx, req.GetHeight())
otherResults, ctx := app.BuildDependenciesAndRunTxs(ctx, otherTxs)
endBlockResp := app.EndBlock(ctx, abci.RequestEndBlock{
```

BuildDependenciesAndRunTxs
sei-chain/app/app.go

```
dependencyDag, err := app.AccessControlKeeper.BuildDependencyDag(ctx,
app.txDecoder, app.GetAnteDepGenerator(), txs)

txResults, ctx = app.ProcessTxs(ctx, txs, dependencyDag,
app.ProcessBlockConcurrent)
```

ProcessTxs
sei-chain/app/app.go

```
concurrentResults, ok := processBlockConcurrentFunction(
txResults := app.ProcessBlockSynchronous(ctx, txs)
```

ProcessBlockConcurrent
sei-chain/app/app.go

```
for txIndex, txBytes := range txs {
    go app.ProcessTxConcurrent(
```

mm

EndBlock
sei-chain/app/abci.go

```
return app.BaseApp.EndBlock(ctx, req)
```

EndBlock
sei-cosmos/baseApp/abci.go

```
res = app.endBlocker(ctx, req)
```

EndBlocker
sei-chain/app/app.go

```
return app.mm.EndBlock(ctx, req)
```

EndBlock
sei-cosmos/types/module/module.go

```
for _, moduleName := range m.OrderEndBlockers {
    module, ok := m.Modules[moduleName].(EndBlockAppModule)
    moduleValUpdates := module.EndBlock(ctx, req)
```

dex

EndBlock
sei-chain/x/dex/module.go

```
newValidContractsInfo, newOutOfRentContractsInfo,
failedContractToReasons, ctx, ok := contract.EndBlockerAtomic(ctx,
&am.keeper, validContractsInfo, am.tracingInfo)
```

EndBlockerAtomic
sei-chain/x/dex/module.go

```
handleDeposits(spanCtx, cachedCtx, env, keeper, tracer)
runner := NewParallelRunner(func(contract types.ContractInfoV2) {
    OrderMatchingRunnable(spanCtx, cachedCtx, env, keeper, contract, tracer)
handleSettlements(spanCtx, cachedCtx, env, keeper, tracer)
handleUnfulfilledMarketOrders(spanCtx, cachedCtx, env, keeper, tracer)
```

OrderMatchingRunnable
sei-chain/x/dex/contract/abci.go

```
} else if settlements, err := HandleExecutionForContract(ctx, sdkContext,
contractInfo, keeper, pairs, orderBooks, tracer); err != nil {
```

HandleExecutionForContract
sei-chain/x/dex/contract/execution.go

```
if err := CallPreExecutionHooks(ctx, sdkCtx, contractAddr, dexkeeper,
registeredPairs, tracer); err != nil {

settlements := ExecutePairsInParallel(sdkCtx, contractAddr, dexkeeper,
registeredPairs, orderBooks)
```

ExecutePairsInParallel
sei-chain/x/dex/contract/execution.go

```
for _, pair := range registeredPairs {
    go func() {
        pairSettlements := ExecutePair(pairCtx, contractAddr, pair, dexkeeper, orderbook)
```

ExecutePair
sei-chain/x/dex/contract/execution.go

```
exchange.AddOutstandingLimitOrdersToOrderbook(ctx, dexkeeper, limitBuys, limitSells)
marketOrderOutcome := matchMarketOrderForPair(ctx, typedContractAddr, pair, orderbook)
limitOrderOutcome := exchange.MatchLimitOrders(ctx, orderbook)
```