

Cette approche maximise la réutilisation des calculs effectués dans la phase d'exécution initiale, et augmente le débit de traitement des transactions.

Nous validons les idées et techniques proposées dans cette thèse sur des évaluations standardisées comme TPC-C, et des versions modifiées de TPC-H et TPC-E, ainsi que sur d'autres micro-évaluations. Nous montrons que l'utilisation de ces techniques multiplie la performance d'un facteur 2 à 100 dans de nombreux cas d'utilisation. De surcroît, en désactivant certaines optimisations du compilateur, nous obtenons une manière systématique et précise d'évaluer leurs contributions individuelles à la performance.

Mots clefs : base de données transactionnelles, analyse de programmes, techniques de compilation, réparation des transactions, programmation générative, optimisation de programmes, programmation fonctionnelle, compilation, staging, spécialisation des structures de données, déforestation, évaluation partielle.

Contents

Acknowledgements	i
Abstract (English/Français)	iii
1 Introduction	1
1.1 Thesis statement	2
1.2 Contributions	4
1.3 Thesis Outline	7
2 Background	9
2.1 Transaction Processing	9
2.2 Parallel Transaction Processing Architecture	10
2.2.1 Shared-Nothing	10
2.2.2 Shared-Everything	12
2.3 Concurrency Control Algorithms	12
2.3.1 Multiversion Concurrency Control	14
2.4 Query Processing Architecture	16
2.5 Query Compilation	17
2.6 Program Transformation via Staging	19
2.7 Applying Optimization Techniques	20
2.7.1 Common Sub-expression Elimination (CSE)	22
2.7.2 Dead Code Elimination (DCE)	22
2.7.3 Combining Optimizations (Speculative Rewriting)	22
2.7.4 Data Structure Optimizations	23

2.7.5	Loop Inversion	23
2.7.6	Loop Fusion and Deforestation	23
2.8	Parallel Query Execution	25
3	Compiling Database Applications and Transaction Programs	27
3.1	Introduction	27
3.2	Architecture	31
3.3	A DSL for Database Applications	34
3.4	Optimizations	37
3.4.1	Removing Intermediate Materializations	37
3.4.2	Automatic Indexing	39
3.4.3	Automatic Indexing Example	40
3.4.4	Hoisting	42
3.4.5	Partial Evaluation	43
3.4.6	Record Structure Specialization	43
3.4.7	Mutable Records	44
3.4.8	Removing Dead Index Updates	45
3.5	Inside-Out Data Structures	45
3.6	Evaluation	47
3.6.1	Performance Model	47
3.6.2	TPC-C Benchmark	48
3.6.3	View Maintenance Triggers for TPC-H	51
3.7	Concurrency Control	54
3.7.1	Concurrency Model	54
3.7.2	Impact on Optimizations	55
3.7.3	Experiments	55
3.7.4	Improving Concurrency Control Algorithms	57
4	Transaction Repair for Multi-Version Concurrency Control	59
4.1	Introduction	59
4.2	MV3C Design	62
4.2.1	OMVCC Overview	62
4.2.2	MV3C Machinery	63
4.2.3	MV3C Execution	67
4.2.4	MV3C Validation and Commit	69
4.2.5	MV3C Repair	72
4.2.6	Serializability	73
4.3	Interoperability with MVCC	74
4.4	Optimizations	74
4.4.1	Attribute-Level Predicate Validation	74
4.4.2	Reusing Previously Read Versions	75
4.4.3	Exclusive Repair	76
4.5	Implementation	76
4.6	Evaluation	78
4.6.1	Rollback vs. Repair	79
4.6.2	Overhead of MV3C	82
4.7	Serializability Proof	83

Contents

4.8	Translating into MV3C DSL	85
4.9	Extended Evaluation	88
4.9.1	TATP Benchmark	89
4.9.2	TPC-C Benchmark	89
4.9.3	The Ripple Effect	89
5	Related Work	91
5.1	Front-end DSLs	91
5.2	Database Compilers	91
5.3	Application-Level Compilation	92
5.4	Multi-Version Concurrency Control	93
5.5	Partial Rollback and Restart	93
5.6	Nested Transactions	94
5.7	Coordination Avoidance	95
5.8	Timestamp Allocation	95
6	Conclusion and Future Work	97
	Bibliography	99
	List of figures	109
	List of tables	111
	Curriculum Vitae	113