

Contents

Part I Background

1	Emerging Parallel Architectures	3
1.1	Parallel Architecture is Dominating the World	3
1.2	Shared Memory Parallel Architecture	4
1.2.1	Multi-core Architecture	4
1.2.2	Multi-socket Multi-core Architecture	5
1.2.3	Asymmetric Multi-core Architecture	6
1.3	Distributed Memory Parallel Architecture	6
1.3.1	Tight-Coupled Distributed Memory Architecture	7
1.3.2	Loose-Coupled Distributed Memory Architecture	7
1.4	Accelerator	8
1.4.1	GPGPU	9
1.4.2	Intel Xeon Phi	10
1.5	Heterogeneous Parallel Architecture	11
1.6	Chapter Highlights	11
	References	12
2	Conventional Task Scheduling Policies	13
2.1	Manual Task Scheduling Policies	14
2.1.1	Message Passing	14
2.1.2	Multi-threading	15
2.2	Automatic Task Scheduling Policies	15
2.2.1	Task Scheduling Policies for Data Parallelism	15
2.2.2	Task Scheduling Policies for Task Parallelism	16
2.3	Parallel Programming Environments	19
2.3.1	Programming Environments for Data Parallelism	19
2.3.2	Programming Environments for Task Parallelism	22

2.4	Problems in Existing Task Scheduling Systems	24
2.5	Chapter Highlights	24
	References	25

Part II Optimized Task Scheduling for Parallel Architectures

3	Work-Stealing for Multi-socket Architecture	29
3.1	Background and Existing Problems	29
3.1.1	The TRICI Problem	30
3.2	Prior Solutions	32
3.2.1	Scalable Locality-Aware Adaptive Work-Stealing (SLAW)	32
3.2.2	Multi-Threaded Shepherds (MTS)	33
3.2.3	Probability Work-Stealing (PWS)	35
3.2.4	Hierarchical Work-Stealing (HWS)	35
3.2.5	CONTROLLED-PDF	36
3.3	Cache-Aware Bi-tier Work-Stealing	37
3.3.1	Solution Overview	37
3.3.2	Design Overview	38
3.4	Cache-Aware Task Graph Partition Policy	40
3.4.1	Full Tree Oriented Partition Policy	40
3.4.2	General Tree Oriented Partition Policy	43
3.5	Bi-tier Work-Stealing Scheduling Policy	47
3.5.1	Work Stealing Algorithm	48
3.5.2	Task Generating Algorithm	49
3.6	Theoretical Time and Space Bounds	51
3.6.1	Theoretical Bounds for Random Work-Stealing	51
3.6.2	Theoretical Bounds for CAB	52
3.7	Implementation Methodology	54
3.7.1	Compiler Support	54
3.7.2	Runtime Support	55
3.8	Evaluation of CAB	55
3.8.1	Performance of CAB-FTO	57
3.8.2	Performance of CAB-GTO	61
3.9	Summary	69
3.9.1	Chapter Highlights	70
	References	71
4	Work-Stealing for NUMA-enabled Architecture	73
4.1	Chapter Organization	73
4.2	Background and Existing Problems	73
4.3	Prior Solutions	75
4.3.1	Random Pushing	76
4.3.2	Cluster-Aware Hierarchical Stealing (CHS)	77

4.3.3	Cluster-Aware Load-Based Stealing (CLS)	77
4.3.4	Cluster-Aware Random Stealing (CRS)	79
4.3.5	TATL	80
4.3.6	NUMALB	82
4.3.7	Offline Technique for Unstructured Parallelism	84
4.4	Design of Locality-Aware Work-Stealing	87
4.5	Load-Balanced Task Allocator	88
4.6	Cache-Friendly Task Graph Partitioner	91
4.6.1	Decide the Initial Partitioning	91
4.6.2	Search for the Optimal Partitioning	92
4.7	Triple-Level Work-Stealing Policy	94
4.8	Theoretical Validation	96
4.9	Implementation Methodology	97
4.10	Performance Evaluation of LAWS	98
4.10.1	Experimental Platforms	98
4.10.2	Performance of LAWS	100
4.10.3	Effectiveness of Cache-Friendly Task Graph Partitioner	103
4.10.4	Scalability of LAWS	104
4.10.5	Overhead of LAWS	107
4.10.6	Applicability of LAWS	108
4.11	Summary	109
4.11.1	Chapter Highlights	109
	References	110
5	Dynamic Load Balancing for Asymmetric Multi-core	
	Architecture	113
5.1	Chapter Organization	113
5.2	Problem Formulation	114
5.3	Existing Solutions	115
5.3.1	Task Snatching Technique	115
5.3.2	CAMP	117
5.3.3	Bias Scheduling	119
5.3.4	Age-Based Scheduling	121
5.3.5	Speed-Based Balancing	123
5.3.6	Scheduling on AMC with Hardware Support	126
5.4	Theoretical Ideal Task Scheduling	126
5.5	A Practical Polynomial Time Solution	127
5.6	Design of Asymmetric-Aware Task Scheduling	129
5.6.1	Processing Flow of AATS	130
5.7	History-Based Task Allocation	131
5.7.1	Build Task Classes	132
5.7.2	Allocate Task Classes to C-Groups	134

5.8	Preference-Based Work-Stealing	136
5.8.1	Scheduling Within a C-Group	136
5.8.2	Scheduling Among C-Groups	137
5.9	Implementation Methodology of AATS	139
5.10	Performance of AATS	140
5.10.1	Experimental Configurations	140
5.10.2	Performance on Emulated Platform	143
5.10.3	Effectiveness of the Preference-Based Work-Stealing	145
5.10.4	Scalability of AATS	146
5.10.5	Integrating Task-Snatching in AATS	148
5.11	Summary	149
5.11.1	Chapter Highlights	150
	References	150
6	Load Balancing for Heterogeneous Parallel Architecture	153
6.1	Background and Existing Problems	153
6.2	Prior Solutions	155
6.2.1	Static Scheduling	155
6.2.2	Quick Scheduling	156
6.2.3	Split Scheduling	158
6.2.4	FinePar	159
6.3	Heterogeneous-Aware Task Scheduling	161
6.4	Comparison of the Scheduling Policies	162
6.5	Performance of Dynamic Scheduling Policies	164
6.5.1	Experimental Setup	164
6.5.2	Performance	165
6.5.3	Effectiveness of Balancing Workload	167
6.5.4	Effectiveness of Predicting the Performance of GPU	168
6.5.5	Impact of Profiling Granularity	168
6.6	Summary	169
6.6.1	Chapter Highlights	169
	References	170
7	MapReduce for Cloud Computing	173
7.1	Introduction to MapReduce	173
7.1.1	Scheduling Policy in MapReduce	174
7.1.2	Adapting to Other Platforms	175
7.1.3	Variations of MapReduce	176
7.1.4	Existing Problem in Heterogeneous Environment	176
7.2	Prior Solutions	177
7.2.1	Least Progress Policy	177
7.2.2	Longest Approximate Time to End Policy	178

7.2.3	Calculating Progress Score	179
7.2.4	Problems in Existing Solutions	180
7.2.5	Tarazu	181
7.3	Self-adaptive MapReduce Scheduling	184
7.3.1	Overview of SAMR	184
7.3.2	Tuning Phase Weights	185
7.3.3	Calculating Progress Score	185
7.3.4	Identifying Straggler Task.	186
7.3.5	Identifying Slow Node	187
7.3.6	Boosting Straggler Task	188
7.4	Implementation of SAMR	189
7.5	Performance Evaluation	190
7.5.1	Experimental Setup	190
7.5.2	Performance.	191
7.5.3	Effectiveness of Speculative Execution and Weight Tuning	192
7.5.4	Parameter Selection in SAMR.	193
7.6	Summary	196
7.6.1	Chapter Highlights	196
	References	197
8	QoS-Aware Task Reordering for Accelerators	199
8.1	Background and Existing Problems	199
8.2	Prior Work on Handling Accelerator Co-location	200
8.2.1	TimeGraph	201
8.2.2	GPU-EvR	202
8.2.3	Simultaneous Multi-kernel (SMK).	204
8.2.4	GPU Thread Preemption.	206
8.3	Real System Investigation on Accelerator Co-location.	206
8.4	Investigation on Priority-Based Scheduling Policy	208
8.5	Design of Task Scheduling Mechanism on Accelerators.	209
8.6	Case Study: QoS-Aware Task Scheduling on Accelerator	210
8.6.1	Root Causes of Long Tail Latency at Co-location	210
8.6.2	Design of Baymax	211
8.7	Task Duration Modeling in Baymax	212
8.7.1	Task Duration Predictor	212
8.7.2	Selecting Representative Features	213
8.7.3	Low Overhead Prediction Models	214
8.7.4	Minimizing Prediction Error	215
8.7.5	Prediction Accuracy	215
8.8	Scheduling Hand-Written Kernels and Library Calls	218
8.8.1	Breaking down the End-to-end Latency	218
8.8.2	Scheduling Policy	219

8.9	Scheduling Data Transfer Tasks	222
8.9.1	Characterizing PCI-e Bandwidth Contention	222
8.9.2	Scheduling Policy	223
8.10	Performance of Baymax	224
8.10.1	Experimental Configuration	224
8.10.2	QoS and Throughput	225
8.10.3	Scheduling Data Transfer Tasks	226
8.10.4	Beyond Pair-Wise Co-locations	227
8.11	Summary	228
8.11.1	Chapter Highlights	229
	References	230
 Part III Summary and Discussion		
9	Summary and Discussion	235
9.1	Guideline of Scheduling Technique Design	235
9.2	Multi-socket Architecture	236
9.3	NUMA-Enabled Multi-socket Architecture	236
9.4	Asymmetric Multi-core Architecture	237
9.5	Heterogeneous CPU+GPU Architecture	238
9.6	Heterogeneous Cloud Platform	238
9.7	Non-preemptive Accelerator Architecture	239
	Glossary	241

Task Scheduling for Multi-core and Parallel
Architectures

Challenges, Solutions and Perspectives

Chen, Q.; Guo, M.

2017, XVIII, 243 p. 107 illus., 73 illus. in color.,

Hardcover

ISBN: 978-981-10-6237-7