1

**PM-1 3 1/2"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT**

PRECISIONSM RESOLUTION TARGETS

End-to-End Scheduling in Hard Real-Time Multiprocessor Systems

by

Reza Etemadi, B.S., M.S.

**A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy**

**Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University,
Ottawa, Ontario, Canada**

August 29, 1996

©1996, Reza Etemadi

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-19371-3

Name Nazir Elamoudi

Dissertation Abstracts International and Masters Abstracts International are arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation or thesis. Enter the corresponding 6-digit code in the spaces provided.

Real-Time Scheduling in Hard
Real-Time Multiprocessor Systems
 Subject Categories

SUBJECT TERM

UMI

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture.....0729
 Art History.....0377
 Games.....0900
 Dance.....0378
 Fine Arts.....0357
 Information Science.....0723
 Journalism.....0391
 Library Science.....0399
 Mass Communications.....0708
 Music.....0413
 Speech Communication.....0459
 Theater.....0465

EDUCATION

General.....0515
 Administration.....0514
 Adult and Continuing.....0516
 Agricultural.....0517
 Art.....0273
 Bilingual and Multicultural.....0282
 Business.....0688
 Community College.....0275
 Curriculum and Instruction.....0727
 Early Childhood.....0518
 Elementary.....0524
 Finance.....0277
 Guidance and Counseling.....0519
 Health.....0480
 Higher.....0745
 History of.....0520
 Home Economics.....0728
 Industrial.....0521
 Language and Literature.....0279
 Mathematics.....0280
 Music.....0522
 Philosophy of.....0998
 Physical.....0523

Psychology.....0525
 Reading.....0535
 Religious.....0527
 Science.....0714
 Secondary.....0533
 Social Sciences.....0534
 Sociology of.....0340
 Special.....0529
 Teacher Training.....0530
 Technology.....0710
 Tests and Measurements.....0288
 Vocational.....0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language.....0679
 General.....0289
 Ancient.....0290
 Linguistics.....0291
 Modern.....0401
 Literature.....0294
 Classical.....0295
 Comparative.....0297
 Medieval.....0298
 Modern.....0316
 African.....0391
 American.....0305
 Asian.....0352
 Canadian (English).....0355
 Canadian (French).....0355
 English.....0311
 Germanic.....0312
 Latin American.....0315
 Middle Eastern.....0313
 Romance.....0314
 Slavic and East European.....0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy.....0422
 Religion.....0318
 General.....0321
 Biblical Studies.....0319
 Clergy.....0320
 History of.....0322
 Philosophy of.....0469
 Theology.....0323

SOCIAL SCIENCES

American Studies.....0323
 Anthropology.....0324
 Archaeology.....0326
 Cultural.....0327
 Business Administration.....0310
 General.....0272
 Accounting.....0770
 Banking.....0454
 Management.....0338
 Marketing.....0385
 Canadian Studies.....0301
 Economics.....0303
 General.....0305
 Agricultural.....0308
 Commerce-Business.....0309
 Finance.....0310
 History.....0511
 Labor.....0358
 Theory.....0366
 Folklore.....0351
 Geography.....0578
 Gerontology.....0578
 History.....0578
 General.....0578

Ancient.....0579
 Medieval.....0581
 Modern.....0582
 Black.....0328
 African.....0331
 Asia, Australia and Oceania.....0332
 Canadian.....0334
 European.....0335
 Latin American.....0336
 Middle Eastern.....0333
 United States.....0337
 History of Science.....0365
 Law.....0398
 Political Science.....0615
 General.....0616
 International Law and Relations.....0617
 Public Administration.....0617
 Recreation.....0814
 Social Work.....0452
 Sociology.....0426
 General.....0427
 Criminology and Penology.....0938
 Demography.....0431
 Ethnic and Racial Studies.....0428
 Individual and Family.....0429
 Studies.....0430
 Industrial and Labor.....0429
 Relations.....0430
 Public and Social Welfare.....0700
 Social Structure and Development.....0344
 Theory and Methods.....0709
 Transportation.....0999
 Urban and Regional Planning.....0453
 Women's Studies.....0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture.....0473
 General.....0285
 Agronomy.....0475
 Animal Culture and Nutrition.....0476
 Animal Pathology.....0359
 Food Science and Technology.....0478
 Forestry and Wildlife.....0479
 Plant Culture.....0480
 Plant Pathology.....0817
 Plant Physiology.....0777
 Range Management.....0746
 Wood Technology.....0306

Biology

General.....0306
 Anatomy.....0287
 Botany.....0308
 Cell.....0309
 Ecology.....0379
 Entomology.....0329
 Genetics.....0353
 Limnology.....0369
 Microbiology.....0793
 Mollusks.....0410
 Mollusks.....0307
 Neuroscience.....0317
 Oceanography.....0416
 Physiology.....0433
 Radiation.....0821
 Veterinary Science.....0778
 Zoology.....0472

Biophysics

General.....0784
 Medical.....0760

EARTH SCIENCES

Biochemistry.....0425
 Geochemistry.....0994

Geology.....0370
 Geology.....0372
 Geophysics.....0373
 Hydrology.....0388
 Mineralogy.....0411
 Paleobotany.....0345
 Paleontology.....0426
 Paleontology.....0418
 Paleontology.....0985
 Paleontology.....0427
 Physical Geography.....0368
 Physical Oceanography.....0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences.....0768
 Health Sciences.....0546
 General.....0300
 Audiology.....0992
 Chemotherapy.....0350
 Dentistry.....0769
 Education.....0758
 Hospital Management.....0982
 Human Development.....0544
 Immunology.....0347
 Medicine and Surgery.....0569
 Mental Health.....0570
 Nursing.....0380
 Nutrition.....0354
 Obstetrics and Gynecology.....0381
 Occupational Health and Therapy.....0371
 Ophthalmology.....0371
 Pathology.....0419
 Pharmacology.....0572
 Pharmacy.....0382
 Physical Therapy.....0573
 Public Health.....0574
 Radiology.....0575
 Recreation.....0575

Speech Pathology.....0460
 Toxicology.....0383
 Home Economics.....0386

PHYSICAL SCIENCES

Pure Sciences.....0485
 Chemistry.....0749
 General.....0486
 Analytical.....0487
 Biochemistry.....0488
 Inorganic.....0738
 Nuclear.....0470
 Organic.....0491
 Pharmaceutical.....0494
 Physical.....0495
 Polymer.....0754
 Radiation.....0405
 Mathematics.....0405
 Physics.....0405
 General.....0986
 Acoustics.....0606
 Astronomy and Astrophysics.....0608
 Atmospheric Science.....0748
 Atomic.....0607
 Electronic and Electricity.....0798
 Elementary Particles and High Energy.....0789
 Fluid and Plasma.....0609
 Molecular.....0610
 Nuclear.....0782
 Optics.....0756
 Radiation.....0411
 Solid State.....0483
 Statistics.....0346
 Applied Sciences.....0984
 Applied Mechanics.....0346
 Computer Science.....0984

Engineering

General.....0537
 Aerospace.....0538
 Agricultural.....0539
 Automotive.....0540
 Biomedical.....0541
 Chemical.....0542
 Civil.....0543
 Electronics and Electrical.....0544
 Heat and Thermodynamics.....0545
 Hydraulic.....0546
 Industrial.....0547
 Marine.....0794
 Materials Science.....0548
 Mechanical.....0743
 Metallurgy.....0551
 Mining.....0552
 Nuclear.....0549
 Packaging.....0765
 Petroleum.....0554
 Sanitary and Municipal.....0554
 System Science.....0554
 Geotechnology.....0428
 Operations Research.....0794
 Plastics Technology.....0795
 Tissue Technology.....0994

PSYCHOLOGY

General.....0421
 Behavioral.....0384
 Clinical.....0422
 Developmental.....0620
 Experimental.....0623
 Industrial.....0424
 Personality.....0625
 Physiological.....0989
 Psychological.....0349
 Psychobiology.....0637
 Psychometrics.....0461
 Social.....0461

The undersigned hereby recommend to the Faculty of Graduate Studies and Research acceptance of the thesis

End-to-End Scheduling in Hard Real-Time Multiprocessor Systems

submitted by Reza Etemadi, B.S., M.S., in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

Chair, Department of Systems
and Computer Engineering

Thesis Supervisor

Thesis Supervisor

External Examiner

Department of System and Computer Engineering

Faculty of Engineering

Carleton University

August 29, 1996

Abstract

An end-to-end approach for scheduling hard real-time periodic transactions in a multiprocessor or distributed system is presented. Each transaction consists of a chain of tasks, where each task is allocated to a processor based on its resource requirements. A transaction may visit a processor several times and it may fork subtransactions and later join them. Assuming fixed priority scheduling on processors, an upper bound for the end-to-end delay of transactions is introduced. The problem is to determine the priority of each task such that the end-to-end deadline for each transaction is met. A priority assignment is feasible if the corresponding upper-bound for each transaction is less than or equal to the actual deadline of the transaction. Heuristic algorithms for priority assignment in a periodic flow-shop model are introduced and their ability to find a feasible schedule are compared to existing heuristic algorithms by using empirical methods. The heuristic algorithms developed for the flow-shop model are extended to support a more complex problem domain (i.e., when revisiting transactions or transactions with subtransactions exist in the system). The effect of revisiting and subtransactions on the relative performance of the extended algorithms is studied through empirical methods. For a real world example the extended algorithms are used to find priorities for different tasks in the signal processing module of a towed array sonar system.

Acknowledgments

I would like to thank Prof. Gerald Karam and Prof. Shikharesh Majumdar, my thesis supervisors, for guiding me through the graduate degree process at Carleton University.

I would like to thank my parents for their encouragement through the challenging and difficult times over the past couple of years.

I would like to thank my beautiful sister for all her support and long distance calls to convince me not to give up.

I would like to thank my wonderful brothers for their concern on my academic achievements.

I would like to thank my friends at Carleton for reminding me time to time that besides sitting in front of computer there are other things to do as well.

I would like to thank Iranian Ministry of Education, Department of System and Computer Engineering, and NSERC for their financial support during my studies.

Table of Contents

1-Introduction.....	1
1.1 Problem Statement.....	1
1.2 The Proposed Solution and Related Work.....	5
1.3 Contributions	8
1.4 Organization.....	9
 2-Background	 10
2.1 Introduction.....	10
2.2 Uniprocessor Scheduling	11
2.2.1 Static Scheduling	11
2.2.2 Dynamic Scheduling.....	17
2.3 Multiprocessor and Distributed Scheduling.....	20
2.3.1 Static Scheduling	21
2.3.2 Dynamic scheduling.....	29
2.4 Summary	30
 3-The System Model and Scheduling Method.....	 32
3.1 System Model	32
3.2 Scheduling Method and Schedulability Criteria.....	34
3.3 Applying the Scheduling Method to more Complex Domains.....	38
3.4 Discussions	48

4-Heuristics For Priority Assignment in the Flow-Shop Model	53
4.1 Introduction.....	53
4.2 Different Priority Assignment Strategies.....	54
4.3 Simulation Model	65
4.4 Simulation Results	66
4.4.1 Simulation Results Without Considering Periodicity.....	67
4.4.2 Simulation Results When Considering Periodicity	80
4.4.3 Summary of Basic Results	86
4.5 The Effect of Workload Parameters	86
4.5.1 The Effect of the Mean of Task Execution Time	87
4.5.2 The Effect of the Coefficient of Variation of Task Execution Time	89
4.5.3 The Effect of the Mean of Transaction Flexibility	89
4.5.4 The Effect of Coefficient of Variation of Transaction Flexibility	94
4.5.5 The Effect of Number of Transactions	94
4.5.6 Summary of Simulation Results	99
 5-Extending Priority Assignment Algorithms to More Complex Problem Domains	 101
5.1 Introduction.....	101
5.2 Extending RTC to More Complex Problem Domains.....	103
5.2.1 Revisiting Transactions.....	103
5.2.2 Transactions with Subtransactions.....	104
5.3 Extending EQF to more Complex Problem Domains.....	107
5.3.1 Revisiting Transactions.....	107
5.3.2 Transactions with Subtransactions.....	108
5.4 Extending WP to more Complex Problem Domains	110

5.4.1	Revisiting Transactions.....	111
5.4.2	Transactions with subtransactions	112
5.5	Simulation Model	116
5.5.1	Simulation Model for Transactions with Subtransactions	116
5.5.2	Simulation Model for Revisiting Transactions.....	118
5.6	Simulation Results	120
5.6.1	Transactions with subtransactions	120
5.6.2	Revisiting Transactions.....	129
5.7	Complexity of Algorithms	138
5.8	Case Study	140
5.9	Summary.....	148
6	Summary and Conclusions	149
6.1	Summary	149
6.2	Summary of Contributions.....	152
6.3	Application Considerations.....	154
6.4	Future Work	154
Appendix A		164
Simulation Results for Algorithm Comparison.....		164

List of Figures

Figure 1.1	Transactions inside Signal Processing of a Sonar System.....	3
Figure 2.1	Precedence Relations Among Tasks of a Transaction in a Processor ..	14
Figure 2.2	Pre-run-time Scheduling	17
Figure 2.3	A Transaction Passing Through Different Processors	21
Figure 3.1	The System Model	33
Figure 3.2	Transactions Passing Through a Processor	34
Figure 3.3	Revisiting in the Model.....	39
Figure 3.4	Computing the Worst-Case Response Time of a Task in the Face of a Revisiting Transaction.....	40
Figure 3.5	Different Ways a Revisiting Transaction Affects the Worst-Case Response Time of other Tasks	41
Figure 3.6	An Example for Computing Worst-Case Response Time in the case of Revisiting Transactions	45
Figure 3.7	Series of Transactions	47
Figure 3.8	Interaction of a Transaction with its Subtransaction.....	48
Figure 4.1	Priority Assignment Using WP.....	63
Figure 4.2	The Performance of EQF Versus LM for a 2 Processor Flow Shop	68
Figure 4.3	The Performance of the EQF Versus LM for a 10 Processor Flow Shop	69

Figure 4.4	PDF of the $(S_{eqf} - S_{lm})$ for 10 Transactions and a 15 Processor Flow Shop.....	71
Figure 4.5	PDF of the $(S_{eqf} - S_{lm})$ for 5 Transactions and a 15 Processor Flow Shop.....	71
Figure 4.6	The Performance of the EQF Versus RTC for a 2 Processor Flow Shop.....	73
Figure 4.7	The Performance of the EQF Versus RTC for a 10 Processor Flow Shop.....	74
Figure 4.8	The Performance of RTC for a 10 Processor Flow Shop when Transactions are Selected for Scheduling Randomly.....	75
Figure 4.9	The Performance of RTC for a 10 Processor Flow Shop when Remaining Slack Time is Divided in Proportion to Tasks Minimum Response Time ...	76
Figure 4.10	Performance of EQF Versus WP for 2 Processor Flow Shop.....	78
Figure 4.11	Performance of EQF Versus WP for 10 Processor Flow Shop.....	78
Figure 4.12	Performance of HOPA Versus WP for 2 Processor Flow Shop.....	79
Figure 4.13	Performance of HOPA Versus WP for 10 Processor Flow Shop.....	79
Figure 4.14	Performance of EQF Versus LM for 3 Processor Periodic flow Shop.....	82
Figure 4.15	Performance of EQF Versus LM for 3 Processor Non-Periodic Flow Shop.....	82

Figure 4.16	Performance of EQF Versus RTC for 3 Processor	
	Periodic Flow Shop.....	83
Figure 4.17	Performance of EQF Versus RTC for 3 processor	
	Non-Periodic Flow Shop.....	83
Figure 4.18	Performance of EQF Versus WP for a 3 Processor	
	Periodic Flow Shop.....	84
Figure 4.19	Performance of EQF Versus WP for a 3 Processor	
	Non-Periodic Flow Shop.....	84
Figure 4.20	Performance of WP Versus HOPA for a 3 Processor	
	Periodic Flow Shop.....	85
Figure 4.21	Performance of WP Versus HOPA for a 3 Processor	
	Non-Periodic Flow Shop.....	85
Figure 4.22	The Effect of CVE on Relative Performance of EQF and WP	90
Figure 4.23	The Effect of CVE on Relative Performance of HOPA and WP	90
Figure 4.24	The Effect of CVE on Relative Performance of EQF and RTC	91
Figure 4.25	The Effect of CVE on Relative Performance of EQF and LM.....	91
Figure 4.26	The Effect of MF on Relative Performance of EQF and WP	92
Figure 4.27	The Effect of MF on Relative Performance of HOPA and WP.....	92
Figure 4.28	The Effect of MF on Relative Performance of EQF and RTC	93
Figure 4.29	The Effect of MF on Relative Performance of EQF and LM	93
Figure 4.30	The Effect of CVF on Relative Performance of EQF and WP	95
Figure 4.31	The Effect of CVF on Relative Performance of HOPA and WP.....	95
Figure 4.32	The Effect of CVF on Relative Performance of EQF and RTC	96
Figure 4.33	The Effect of CVF on Relative Performance of EQF and LM	96
Figure 4.34	The Effect of NT on Relative Performance of EQF and WP	97
Figure 4.35	The Effect of NT on Relative Performance of HOPA and WP	97

Figure 4.36	The Effect of NT on Relative Performance of EQF and KTC.....	98
Figure 4.37	The Effect of NT on Relative Performance of EQF and LM	98
Figure 5.1	Transaction with Subtransactions	102
Figure 5.2	Revisiting Transaction	102
Figure 5.3	Simulation Model for Transactions with Subtransactions.....	117
Figure 5.4	The Effect of Subtransactions for EEQF and EWP (CVE=0.25 CVF=0.0).....	123
Figure 5.5	The Effect of Subtransactions for EEQF and EWP (CVE=0.25 CVF=0.5).....	123
Figure 5.6	The Effect of Subtransactions for EEQF and EWP (CVE=1.0 CVF=0.0).....	124
Figure 5.7	The Effect of Subtransactions for EEQF and EWP (CVE=1.0 CVF=0.5).....	124
Figure 5.8	The Effect of Subtransactions for EEQF and EWP (CVE=2.0 CVF=0.0).....	125
Figure 5.9	The Effect of Subtransactions for EEQF and EWP (CVE=2.0 CVF=0.5).....	125
Figure 5.10	The Effect of Subtransactions for ERTC and EWP (CVE=0.25 CVF=0.0).....	126
Figure 5.11	The Effect of Subtransactions for ERTC and EWP (CVE=0.25 CVF=0.5).....	126
Figure 5.12	The Effect of Subtransactions for ERTC and EWP (CVE=1.0 CVF=0.0).....	127
Figure 5.13	The Effect of Subtransactions for ERTC and EWP (CVE=1.0 CVF=0.5).....	127

Figure 5.14	The Effect of Subtransactions for ERTC and EWP (CVE=2.0 CVF=0.0).....	128
Figure 5.15	The Effect of Subtransactions for ERTC and EWP (CVE=2.0 CVF=0.5).....	128
Figure 5.16	The Effect of Revisiting (CVE=0.5 CVF=0.0 MF=3.0).....	131
Figure 5.17	The Effect of Revisiting (CVE=0.5 CVF=0.5 MF=3.0).....	131
Figure 5.18	The Effect of Revisiting (CVE=1.0 CVF=0.0 MF=3.0).....	132
Figure 5.19	The Effect of Revisiting (CVE=1.0 CVF=0.5 MF=3.0).....	132
Figure 5.20	The Effect of Revisiting (CVE=0.5 CVF=0.0 MF=2.0).....	135
Figure 5.21	The Effect of Revisiting (CVE=0.5 CVF=0.5 MF=2.0).....	135
Figure 5.22	The Effect of Revisiting (CVE=1.0 CVF=0.0 MF=2.0).....	136
Figure 5.23	The Effect of Revisiting (CVE=1.0 CVF=0.5 MF=2.0).....	136
Figure 5.24	Transactions inside Signal Processing of a Sonar System.....	142
Figure 6.1	A Problem with EQF and WP.....	155

List of Tables

Table 3.1	Algorithm <i>RELEASE</i>	44
Table 3.2	Algorithm <i>LOAD</i>	44
Table 3.3	Workload for the Example of Figure 3.6	45
Table 4.1	A Transaction Set.....	57
Table 4.2	Deadline Assignment Using LM Algorithm.....	57
Table 4.3	Deadline Assignment Using EQF Algorithm	59
Table 4.4	Deadline Assignment Using RTC.....	60
Table 4.5	The Effect of Local Tasks in the EQF Algorithm.....	72
Table 4.6	Experiments for Flow Shop	88
Table 5.1	Experiments for Transactions with Subtransactions.....	118
Table 5.2	Experiments For The Revisiting	120
Table 5.3	Distribution of Problem Instances when CVE=0.5, CVF=0.0, MF=3.0 NT=5, NP=4, and NJ=7.....	133
Table 5.4	Distribution of Problem Instances When CVE=0.5, CVF=0.0, MF=2.0, NT=5, NP=4, and NJ=4.....	137
Table 5.5	CPU Time for Comparing the Algorithms.....	140
Table 5.6	Task Execution Times, Periods, and Deadlines for Signal Processing Module of the Towed Array Sonar System.....	143
Table 5.7	Task's Priorities for Scenario 1.....	146
Table 5.8	Tasks's Priorities for Scenario 2	147

Glossary of Acronyms and Symbols

LM	The laxity monotonic algorithm
EQF	The equal flexibility algorithm
RTC	The real-time channel algorithm
WP	The worst position algorithm
HOPA	The optimized heuristic priority assignment
EEQF	The extended EQF algorithm (transactions with subtransactions or revisiting transactions)
EWP	The extended WP algorithm (transactions with subtransactions or revisiting transactions)
ERTC	The extended RTC algorithm (transactions with subtransactions)
RDC	Raw data computation
BBDE	Broadband and demon computation
NB	Narrowband computation
TP	Tracking parameter control
DS	Display computation
TC	Tracking computation
B	Beamforming computation
A	Audio computation
PC	Parameter change
ME	Mean of task execution time
MF	Mean of transaction flexibility

CVE	Coefficient of variation of task execution time
CVF	Coefficient of variation of transaction flexibility
NT	Number of transaction
NJ	Number of tasks in a revisiting transaction
NS	Number of stages
NP	Number of processors
DEADLINE	The current deadline of a transaction or a subtransaction
WORK	The current remaining work of a transaction or a subtransaction
HRT	Highest response time
WCET	The worst-case execution time
WCRT	The worst-case response time
S_{eqf}	Remaining slack time in EQF algorithm
S_{lm}	Remaining slack time in LM algorithm
T_i	The task i
t_i	The transaction i
$f(t_i)$	The flexibility of transaction i
$\{t_i\}$	The set of processors traversed by transaction i
P_i	The processor i
Q_i	The period of task i
D_i	The deadline of task i
C_i	The worst-case execution time of task i
B_i	The worst-case blocking time of task i due to the operation of a concurrency control protocol
R_i	The worst-case response time of task i when there is no blocking due to the operation of a concurrency control protocol

R_i^b	The worst-case response time of task i when there is blocking due to the operation of a concurrency control protocol
R_i^{b+}	An upper-bound for the worst-case response time of task i when there is blocking due to the operation of a concurrency control protocol
$T_{ij}(k)$	The k^{th} activation of task belonging to transaction i and running on processor j
T_{ij}	The task belonging to transaction i and running on processor j
$T_{z,r,j}$	The task corresponding to the r^{th} revisit of transaction z on processor j
R_{ij}	The worst-case response time of the task running on processor j and belonging to transaction i
R_{ij}^b	The worst-case response time of the task running on processor j and belonging to transaction i when there is blocking due to the operation of a concurrency control protocol
C_{ij}	The worst-case execution time of the task running on processor j and belonging to transaction i
$C_{z,r,j}$	The worst-case execution time the task corresponding to the r^{th} revisits of transaction z on processor j
$hp(l)$	The set of all tasks running on the same processor as task l and with priority higher than task l
$hptask(T_i)$	The set of all tasks running on the same processor as T_i with priority higher than T_i and not belonging to any revisiting transactions
$hptrans(T_i)$	The set of all revisiting transaction having at least one task with priority higher than task i except the one to which task i belongs

$Next^k(T_{z,r,j})$	The task corresponding to the k^{th} task following task $T_{z,r,j}$ in transaction z
$PRI(T_i)$	The priority of task i
$PRI(T_{ij})$	The priority of task belonging to transaction i and running on processor j
$PR(T_i)$	The worst-case execution time of task i
$PR(T_{ij})$	The worst-case execution time of task belonging to transaction i and running on processor j
$dl(T_{ij})$	The local deadline of a task running on processor j and belonging to transaction i
$dl(t_i)_j$	The deadline of transaction i at stage j
$unschedule(j)$	The set of unscheduled tasks on processor j
N	The total number of tasks in the system
K	The number of processors in a parallel stage
m	The number of processors in a flow shop

Chapter 1

Introduction

1.1 Problem Statement

With the ever increasing demands for real-time applications, the next generation of real-time systems will be dynamic, large and complex [34], operating in environments that can be constantly changing. An important class of such systems is called *hard real-time systems*. The correctness of hard real-time systems depends not only on the functional results of the computation, but also on the *timeliness* of the results; i.e., failure to meet timing requirements is a failure in the system specification. In such environments, *analysability* (e.g., the ability to determine whether an application's computations can meet the timing requirements) needs to be provided in the face of unpredictable task arrivals and asynchronous concurrent sharing of system resources.

Examples of these real-time applications include real-time communications, and signal processing functions typical of sonar, radar, and similar data processing systems. A common feature of these real-time applications is that timing constraints are associated with many of the application activities. The problem of designing and analysing real-time systems is further complicated because many real-time applications demand non-homogenous multiprocessor systems to meet their requirements.

One of the most critical issues and a topic of much current research, is how to develop real-time systems such that the timing behavior can be analysed knowing task interactions, task resource requirements and environment behavior.

This research considers real-time systems that are characterized by a set of transactions; each transaction is composed of a set of communicating tasks with resource requirements and precedence relations. Transactions are triggered either in the environment or inside the system and eventually they send responses to the environment or terminate somewhere inside the system. A transaction may fork sub-transactions and wait somewhere in the system for their completion, before continuing execution. Transactions may also visit a resource several times before their completion and they may compete to access a non-preemptable shared resource. Transactions may have end-to-end timing constraints and are characterized by a minimum interarrival time for each transaction. That is, transactions are generated either periodically or sporadically.

In this model it is assumed that each stimulus triggers a set of precedence related tasks (i.e., a transaction) inside the system. Such a behavior is the direct consequence of *the functional requirements of the system*. For example in a sonar system, a signal must be sampled, processed and displayed within a certain time interval. However, the signal may pass through several processing elements, each processing element suitable for a specific computation. The sampling of the signal can be done by a dedicated interrupt processor. Processing of the signal is then carried out by one or more special or general purpose processors and eventually by another console processor that controls the output display. Figure 1.1 illustrates a sonar signal processing component based on the design of a real towed array sonar system. It shows several transactions using various tasks to perform different signal processing functions such as beamforming, tracking, as well as control and data management. This system is discussed in detail in Chapter 5. Another example could be from the area of communication networks. Real time channels with defined end-to-end deadlines are desired in a multi-hop network. Different channels might have different timing requirements and packets belonging to different channels should be scheduled at each node such that the overall scheduling guarantees the end-to-end timing constraints for each stream.

Figure 1.1 Transactions inside Signal Processing of a Sonar System

The stage at which a performance guarantee is done, distinguishes static and dynamic real-time systems. In static systems, a guarantee is provided during design because the application has predefined requirements. In this research, although the major concern is static systems, on-line creation and termination of transactions could also be added. In many real-time applications, transactions arrive on the system aperiodically and they need to be scheduled on-line. If no feasible schedule is found for them, the applications responsible for those transactions should be notified. Sometimes, a transaction with a given deadline and period is established on-line and terminated later. The scheduling method proposed in this research may also be suitable for handling such on-line transactions.

Many factors may affect the analysability of the system. These include the scheduling paradigm, the complexity of task interactions, and the environment behavior. However, analysability is not the sole objective. Besides analysability, it is desired that:

- the run-time system supports the assumptions made in the scheduling method in a precise manner;
- the system resources are not underutilized because of assumptions made for making the system analysable;
- the units of scheduling during run-time are software components defined during the design phase of the system.

In summary, the problem is to find a scheduling method that makes the real-time system analysable for end-to-end deadline and that supports the above requirements for the suggested application model.

1.2 The Proposed Solution and Related Work

As stated earlier, the major challenge in real-time systems is to make the system analysable such that the temporal behavior of the system can be predicted. This prediction is based on either statistical measures or deterministic ones. In this research, we are interested mainly in deterministic measures because: (1) some real time applications require hard deadlines; and, (2) it makes the system analysable under complex real-time models and different scheduling methods. With regard to statistical guarantees, many of the researchers have assumed FIFO scheduling at each node. More complicated scheduling mechanisms are computationally difficult, if not intractable [23]. Although the FIFO discipline is one of the simplest scheduling mechanisms, it is generally considered ineffective for real-time applications [23]. On the other hand, many real-time applications need a statistical guarantee for a given time interval which makes the system more difficult to analyze. Real-time voice transmission is one example. There are two problems with providing such guarantees. First, it is not clear if one can obtain, a priori, an exact stochastic characterization of the traffic entering the real-time system; and second, even if such a characterization is available it is relatively hard to characterize the traffic within the system.

A considerable amount of research has been done on providing deterministic performance guarantees for real-time systems. For this thesis only work that investigates end-to-end deterministic performance guarantees for multiprocessor systems is surveyed.

End-to-end performance guarantees have been considered by other researchers as well. The Spring project [35] considers deterministic end-to-end performance guarantees for safety critical periodic tasks. A task consists of a chain of subtasks with precedence relations and resource requirements. Static scheduling is done to guarantee the end-to-end performance requirement of these tasks. Dynamic tasks are also considered in this project. These tasks are guaranteed on-line and they might have a periodic traffic pattern. The cen-

tral concept in the Spring project is the generation of an explicit execution plan for the subtasks. The Mars project [18] follows the same concept of scheduling as the Spring project except that it does not consider dynamic task arrivals.

The scheduling method chosen in this research is closely related to the method proposed by Kandlur [17] and Zheng [44]. Kandlur and Zheng use a method to schedule real-time channels on different nodes of a multi-hop network. The scheduling method is based on independent scheduling for each processor. That is, each processor is considered separately and tasks allocated to it are scheduled such that the combination of these separate schedules satisfies the timing requirements of all transactions. To make separate scheduling possible, the traffic pattern going through each processor should be known. Assuming periodic and sporadic events, there should be a minimum interval between two consecutive invocations of a task on a processor. This can be done by the kernel or application tasks on each processor for every task during run-time. In this research, we apply this scheduling method to the other real-time applications which require more complicated models. Real-time channels considered by Kandlur and Zheng pass through nodes from source to destination and there is no concept of looping or forking and joining of channels.

Using fixed priority scheduling, the worst-case response time of each transaction on each processor can be computed and consequently an upper-bound for the end-to-end delay of each transaction can be found. The upper-bound for various kinds of transactions is formulated. The method for computing the upper-bound for *simple transactions* (transactions that do not revisit a processor or create subtransactions) is the same as the one introduced in [3]. The computation of an upperbound on delay for *complex transactions* is one of the contributions of this thesis.

To make independent scheduling practical, two major problems must be considered. These problems are:

- *The allocation problem.* This problem is concerned with allocation of tasks to processors. The question is how tasks should be assigned to processors such that the assignment increases schedulability.
- *The priority assignment problem.* After allocating tasks to processors, we need algorithms to assign priorities to tasks such that the end-to-end deadlines of transactions are satisfied.

In this research, we are only concerned about the second problem and it is assumed that allocation of tasks to processors has been done. Note that, one aspect of analysis is the guarantee that the system can be designed so that all transactions meet their end-to-end deadlines. Other forms of analysis may be of interest in some applications, such as, guarantee about jitter [39] and use of memory resources; however, these are beyond the scope of this research.

The problem of assigning priority to transactions is *NP-hard* [3] and heuristic algorithms are used for assigning priorities. If a feasible priority assignment is found then it is guaranteed that every invocation of each transaction will meet its deadline.

The major challenge for this research is finding good heuristics for assigning priorities. A good heuristic is very likely to find a feasible schedule when one exists. Different heuristic methods are proposed and their performance is compared using empirical methods. For comparing the performance of different algorithms, a random set of problem instances is generated. Running the algorithms over the set determines which algorithm is able to find more feasible solutions. Random model generation is used in this research mainly to identify weaknesses in an algorithm and to test the sensitivity of algorithms to parameter settings.

Current heuristics [3, 12] find some feasible solutions in a subset of the suggested application model (i.e., a flow shop: when the model consists only of simple transactions). The objectives of this research are: (1) to develop heuristics to generate more feasible solutions for the simple transaction model; and, (2) to develop heuristics for computing feasible solutions for more complicated transaction models (i.e., when revisiting transactions or transactions with subtransactions also exist in the model).

1.3 Contributions

The primary research contribution of this thesis is developing and characterizing different heuristic algorithms for the priority assignment problem. The novel work explored in the thesis area is described below.

- A problem with the LM (laxity monotonic) algorithm [3] is identified and a new heuristic algorithm EQF (equal flexibility) that alleviates the problem is introduced. Using empirical methods, the performance of the algorithms are compared.
- The EQF algorithm, however, has its own drawbacks; some of the problems with EQF are identified. To overcome some of these drawbacks, the WP (worst position) algorithm is introduced. The performance of WP versus EQF is studied. The drawbacks of WP are also discussed.
- The performance of the RTC (real-time channel) algorithm is compared to EQF. RTC is originally proposed for establishing real-time channels [17] and it is used in this research with some modifications. RTC is suitable for creating dynamic transactions.
- The performance of WP is compared to the HOPA (Heuristic Algorithm for Optimized Priority Assignment) algorithm [12].

- EQF, WP, and RTC are extended to solve complex transactions (i.e, revisiting transactions and transactions with subtransactions) and the performance of the extended algorithms is compared.
- Through simulation, the effect of the workload parameters on the relative performance of algorithms is studied and substantial insights into the behaviour of algorithms is gained.
- An algorithm to compute an upperbound on the end-to-end response time of revisiting transactions is developed.

1.4 Organization

The rest of this dissertation is organized as follows. In Chapter 2, a survey of the existing work on real-time scheduling is presented. Chapter 3 describes the scheduling method used to make the real-time system analysable. This method gives a bound on end-to-end latency of each transaction. The priority assignment problem for a simple model (a flow-shop) is discussed in Chapter 4 where different heuristics are presented and their performance is compared through simulation. In Chapter 5, the algorithms introduced in Chapter 4 are extended for handling complex transactions and the performance of the extended algorithms is compared. In this chapter the extended algorithms are used to assign priorities to different tasks in the signal processing module of a towed array passive sonar system. We conclude in Chapter 6 by noting the conclusions and suggestions for future work.

Chapter 2

Background

2.1 Introduction

Scheduling is one of the major concerns in the design of analyseable real-time systems. Scheduling a set of *transactions* consists of sequencing them on one or more processors such that the utilization of resources optimizes some scheduling criteria. In real-time systems the scheduling criteria is meeting transaction deadlines. The scheduler should consider a set of constraints. Constraints may come from the design phase (e.g., precedence and exclusion relations among tasks as well as from resource requirements), timing requirements of the system for responding to events, and from the behavior of the environment.

The scheduling paradigm plays a major role in the way the system should be analyzed and the system model that can be supported. In the literature, several different paradigms have been introduced. These paradigms for uniprocessor and multiprocessor system are reviewed and the advantages and disadvantages of each are discussed in this chapter.

Paradigms are reviewed under the sub-headings of static and dynamic scheduling. For static scheduling, only the cases with deterministic performance guarantees are discussed. In deterministic cases, it is guaranteed that all the task invocations will meet their deadlines. On the other hand, in statistical cases, it is guaranteed that not more than a specific percentage of task invocations, on average, or during a specified time interval, will miss their deadlines. In the case of dynamic scheduling no guarantee is made a priori and guarantee is made for each task invocation during run-time.

2.2 Uniprocessor Scheduling

2.2.1 Static Scheduling

Static scheduling requires the complete a priori knowledge of all task characteristics and of the environment behaviour. In static scheduling if the design is changed (e.g., adding another task to the system or changing task interactions) or a parameter in the environment is changed (e.g., changing the period of a periodic event) then it is necessary to compute a new schedule. Although this computation is off-line, it is important that it be tractable and a new schedule be found in a reasonable amount of time. In static scheduling, sometimes processor utilization must be sacrificed to reach the performance goals and make off-line scheduling tractable. In static scheduling the behaviour of the environment should be deterministic or at least be modeled deterministically. Because of nondeterminism in the behaviour of the environment, scheduling might be done for the worst-case situation which could degrade the processor utilization considerably. In this section only periodic and sporadic events are considered. A static scheduling algorithm is said to be optimal if for any set of tasks, it finds a feasible schedule whenever there is one.

The most important approaches concerning static scheduling in uniprocessor-based real-time systems are reviewed in this section.

Approach#1: Assigning static priorities to tasks, and during run-time, allocating system resources to the task that has highest priority among all other tasks.

Liu and Layland [21] have studied fixed-priority scheduling of a synchronous task set (in a synchronous task set the release time is the same for all tasks) on one processor under the assumption that $D_i = Q_i$ (i.e., deadline of task is equal to its period) for all tasks. They showed that an optimal fixed-priority algorithm can be obtained by assigning priorities to tasks according to rate-monotonic priority assignment; that is the task with shortest period

has the highest priority. However, the rate-monotonic assignment is not optimal if $D_i \neq Q_i$ for some of the tasks. It is shown that the deadline monotonic priority assignment is an optimal fixed priority algorithm for a synchronous task set when the deadlines of tasks is less than or equal to their periods [19]. It is interesting to note that the deadline monotonic priority assignment reduces to rate-monotonic if $D_i = Q_i$. Liu and Layland [21] gave characterizations which lead to an algorithm to decide whether or not the schedule produced by a particular priority assignment is valid. Their characterizations are also valid when $D_i \neq Q_i$. They described this result as follows: *A critical instant for any task occurs whenever the task is requested simultaneously with all higher priority tasks. A critical instant of a task is defined to be an instant at which the maximum response time will occur.*

What is important in the above result is that it suggests a method for determining if the schedule produced by a particular priority assignment is valid. This result provides a way to do scheduling deterministically. For synchronous systems all tasks make their first request at time zero; hence according to the above result the schedule produced by a particular priority assignment is valid if and only if the deadline of the first request of each task is met. Also the above result motivates interest in synchronous systems because it is the worst-case. In [19], it is shown that the problem of finding a feasible priority assignment for an asynchronous task set is NP hard.

In the previous discussion, tasks are assumed to be independent and preemptable. For dependent tasks, there is a possibility that unbounded priority inversion happens: a high priority task is ready to execute but is blocked by the execution of a lower priority task because the lower priority task is accessing a non-preemptable shared resource. The amount of time the lower priority task stays in the critical section could be unbounded and consequently the high priority task confronts unbounded priority inversion. One of the

solutions proposed for this dilemma is priority inheritance [28], in which a task that blocks a high priority task inherits the priority of the task it is blocking for the duration of the blocking period. This solution prevents unbounded periods of priority inversion.

In the literature [40], some flaws have been reported for static priority assignment. They are briefly described.

- Static priority-driven schemes are only capable of producing a very limited subset of possible schedules for a given set of tasks; so it is not optimal. In other words there may be a schedule but it can not be determined by static priority assignment.
- Because priority driven schemes can produce only a very limited subset of all possible schedules, when complex relations between processes and resource sharing constraints must be taken into account, the level of processor utilization required to guarantee schedulability may be too low to be really useful in practice.
- Generally, the priority assignment model does not define a tight timing constraint on the actual completion time of a task, other than ensuring that it completes before its deadline. As a result, there may be a significant jitter generated by the priority assignment model, especially for lower priority tasks.

Static priority assignment has some advantages which are listed below.

- In fixed priority assignment, it is possible to predict which tasks will miss their deadlines when overload occurs.
- Scheduling transactions consisting of interrelated tasks can be done in the same way as for a transaction consisting of a single task. For example, transaction t_i in Figure 2.1 is equivalent to a transaction consisting of single task with execution time equal to the

sum of execution time for tasks $T1$, $T2$, $T3$ and $T4$. The equivalent transaction has the same period and deadline as the transaction t_j . It is assumed that the priority of tasks $T1, T2, T3$ and $T4$ is same as the priority assigned to transaction t_j .

Figure 2.1 Precedence Relations Among Tasks of a Transaction in a Processor

Approach#2: Assigning dynamic priorities to tasks

In this approach, the priority of a task can change from request to request. Liu and Layland [21] showed that the earliest deadline first algorithm (EDF) is optimal in a single-processor system in which all tasks are independent, periodic and preemptable at any time. It is optimal in the sense that it can always find a feasible schedule if one exists and consequently it can fully utilize the processor. Systems using the EDF algorithm always execute the task with the earliest deadline. Liu and Layland [21] have also derived the following necessary and sufficient condition for the schedulability of a system with n tasks. This condition is valid for the case in which the period of each task is equal to its deadline.

$$\frac{C_1}{Q_1} + \frac{C_2}{Q_2} + \dots + \frac{C_n}{Q_n} \leq 1$$

In the above equation C_i and Q_i are the execution-time and period of task T_i , respectively. For the case that the deadline of a task is not equal to its period, Zheng [44] has proposed a schedulability analysis that gives a necessary and sufficient condition for a task set to be schedulable. The algorithm presented by Zheng becomes computationally more expensive as processor utilization increases.

Several synchronization protocols have been proposed when priorities are dynamic. Chen et al [4] have extended the original priority ceiling protocol to one which is able to handle dynamic priorities. Jeffay [16] developed schedulability conditions for a set of sporadic tasks that consists of a sequences of phases, each of which may require access to at most one shared resource. He shows that optimal synchronization and scheduling disciplines exist for restricted patterns of resource usage.

Pingali [25] compares the occurrence of preemptions under static and dynamic priority assignment. He shows that the number of preemptions at the resource using static priority assignment is larger than the number of preemptions that would be obtained if dynamic priority assignment is used. He shows that the preemption overhead under dynamic priority assignment is not worse than static assignment.

The EDF algorithm is optimal under normal (non-overload) condition, however, under overload, it is observed that it performs worse even in comparison with random scheduling [13].

Approach#3: Finding an off-line schedule using task deadlines, release-times, and precedence and exclusion relations among segments.

In [41], Xu and Parnas have defined a problem as follows: "We are given a set of tasks, where each task consists of a sequence of segments. Each segment is required to precede a given set of other segments. Each segment also excludes a given set of other segments

(i.e., once a segment has started, its computation can not be preempted by any segment in the set that it excludes). For each task we are given a release-time, a computation-time and a deadline. It is also assumed that we know the computation time and start time of each segment relative to the beginning of the task containing the segment. Our problem is to find a schedule on a single processor for the given set of tasks such that each task starts its execution after its release time and completes its computation before its deadline, and all the precedence and exclusion relations on segments are satisfied".

The problem as stated above is NP-hard even for a special case where each task is composed of a single segment that excludes all other single segments. The branch-and-bound method is used to find the schedule. Figure 2.2 shows the major concepts behind this kind of deterministic scheduling.

The algorithm proposed by Parnas and Xu can also support periodic and sporadic tasks, because:

- 1) if a feasible schedule could be found for the instances of a given set of periodic processes within a time interval between 0 and the least common multiple (LCM) of the process periods, then the given set of periodic processes can be scheduled by repeating the feasible schedule [20]; and,
- 2) it is possible to transform a sporadic process to an equivalent periodic process [22].

The major advantage of the scheduling method proposed by Xu and Parnas is its optimality. It is optimal in the sense that under all constraints such as deadlines, mutual exclusions and precedence relations the algorithm is able to find a feasible schedule when there is one. The algorithm does an exhaustive search to find the proper schedule. Also, because of the determinism in the schedule throughout the entire application execution, it exhibits very low jitter in the execution of each periodic process. Controlling jitter is quite impor-

tant for certain classes of real-time systems such as feedback control systems, for which the stability of the basic transfer function depends on the tight control of jitter in the feedback loop.

Figure 2.2 Pre-run-time Scheduling

2.2.2 Dynamic Scheduling

A dynamic scheduling approach determines schedules for tasks at run-time. Guarantee for meeting a deadline for a task invocation is made at the time of the task invocation. The advantage of this approach is that there is no need to calculate a new schedule for the system after changing the design or changing a parameter in the environment (e.g., decreasing or increasing the period of a periodic external event). In cases where there is non-deterministic behaviour in the environment, dynamic scheduling is often more suitable compared to static scheduling.

In dynamic scheduling, a newly arriving task is said to be schedulable, if it can be scheduled to meet its deadline such that all previously scheduled tasks also remain schedulable.

A dynamic scheduler is said to be optimal if it constructs a feasible schedule for every task system that is feasible. Dertouzos [8] showed that both earliest due date (EDD) and least laxity first (LLF) scheduling algorithms are optimal for the single processor case. He assumed independent tasks.

The above considerations have revealed that the earliest deadline first algorithm is the best one to use in a general multitasking environment, provided that the tasks are preemptable at any arbitrary point in time. Unfortunately, this precondition is not very realistic, since it is only fulfilled by pure computing tasks that are fully independent of each other. In general, however, tasks have resource requirements and, therefore, execute critical regions to lock peripherals and other resources for exclusive and uninterrupted access. Since, no optimal off-line scheduler with polynomial time complexity is known for this case, it is very unlikely that there exists an optimal dynamic scheduler when resources must be shared.

A major performance metric for a dynamic scheduling algorithm is the guarantee ratio, which is the total number of tasks guaranteed versus the total number of tasks that arrive. An optimal dynamic algorithm maximizes the guarantee ratio of tasks that arrive in a system.

In dynamic scheduling, the application may need to query the ability of the scheduler to guarantee a timing requirement because it may affect the future course of the computation.

As an example, in an application, we may have statement of the form

<i>within deadline (d)</i> <i>statement1</i> <i>else</i> <i>statement2</i>

When the execution reach to the *within deadline* statement, the scheduler is called to check whether there are enough resources to execute *statement1* in *d* time units. If enough

resources do not exist *statement2* will be executed. In contrast, in static scheduling, the application is not involved in the scheduling process.

For some real-time applications it is required to guarantee on-line that events can meet their timing requirements. For these applications the overhead of on-line scheduling can be quite high and may make the approach unpractical. This is especially true for complex real-time systems in which an event may trigger a chain of activities inside the system with many interactions and resource requirements. The situation becomes much worse for multiprocessor and distributed systems as will be discussed later.

Work on dynamic scheduling in a uniprocessor system is reported by Chetto [5], Schwan [32], Tia [37], and Thuel [30]. In all this work, it is assumed that periodic tasks exist in the system and dynamic tasks are scheduled jointly with periodic tasks. Chetto, Schwan and Tia assume an *earliest deadline* algorithm, which assigns priorities to tasks according to the proximity to their deadlines. Thuel assumes fixed priority assignment for tasks.

Chetto's algorithm has certain shortcomings. First, it is assumed that periodic task deadlines are equal to their periods, and second, the algorithm assumes that when a new dynamic task arrives, all dynamic tasks previously accepted have completed their execution.

Schwan has relaxed the assumption made by Chetto and has proposed an algorithm that does not give preferential treatment to the periodic tasks. Every task is subject to an acceptance test upon arrival. Unfortunately, this results in undesirable implementation overheads if the real-time workload is mainly periodic.

Tia proposes an optimal acceptance test for scheduling aperiodic requests together with periodic tasks on an earliest-deadline-first-basis. The deadlines of periodic tasks must be met. Aperiodic requests are accepted only if their deadlines can also be met. In Tia's

acceptance test only sporadic requests, and not periodic requests, are subjected to test; this is the major difference between Schwan's and Tia's acceptance tests.

Thuel assumes fixed priority scheduling. Periodic tasks are tested for schedulability before run-time so there is no need to run an acceptance test on them at run-time. At run time, they are scheduled according to a fixed priority algorithm and their deadlines are guaranteed to be met. The aperiodic tasks, on the other hand, are tested for schedulability at run-time. They are submitted to an acceptance test at run-time to determine: (1) if their deadlines can be met and, (2) to see if they will cause any guaranteed periodic tasks to miss their deadlines. When a new aperiodic task arrives, an algorithm is run and it tries to find a new priority assignment for the previously accepted tasks (periodic and aperiodic) plus the new aperiodic task.

2.3 Multiprocessor and Distributed Scheduling

The potential speed-up, reliability, and also the requirement for heterogenous processing in some systems are the major reasons for using multiprocessor systems. There are at least three important issues in multiprocessor scheduling:

- processors may have different characteristics;
- different tasks of a transaction may be executed on different processors; and,
- the performance metric is meeting the deadline of each arbitrary transaction. A transaction is triggered either by an internal or external event and in general it consists of interacting tasks. Figure 2.3 shows a transaction triggered by an external event *E*!

The problem of multiprocessor scheduling for meeting transaction deadlines in general is as follows:

Given a set of transactions such that each transaction consists of a set of interrelated tasks, how should tasks be allocated and scheduled on a given number of processors to satisfy the following set of constraints: (1) meeting the timing constraint of each transaction; and, (2) satisfying resource constraints for each task.

Figure 2.3 A Transaction Passing Through Different Processors

Determining a schedule that meets the timing constraints while satisfying the other constraints is the goal of algorithms for multiprocessor scheduling in real-time systems.

2.3.1 Static Scheduling

Approach#1: Tasks are statically bound to processors and static priority assignment is used on each processor.

If we have no restriction on assigning tasks to processors and all processors are identical then we could partition tasks into groups so that each group of tasks can be feasibly scheduled on a single processor according to some scheduling algorithm. The problem is thus reduced to determining a partition of given set of tasks so that, with respect to the designated algorithm to be used for the groups in the partition, the number of processors is minimized. Since for a finite set of tasks there is only a finite number of possible partitions, this problem can be solved by an exhaustive search. However, the number of possible partitions increases exponentially with an increase in the number of tasks. Such an exhaustive

search requires considerable computation time and will offset the advantage gained by using an optimal partition. Hence, it is interesting to consider some good partitioning scheme. By a good partitioning scheme, we mean a scheme according to which the number of processors required is reasonably close to the minimum number of processor needed. In [9] some heuristics for partitioning a set of tasks are introduced.

How should shared resources be handled under this concept? Again, there is the notion of blocking; i.e., the amount of time that a task has to wait in addition to the situation where there is no resource sharing. In the case of multiprocessor environments, a task may also have to wait for a task running on another processor to release the shared resource. Thus, in multiprocessor systems, blocking can be caused by any task of higher, equal or lower priority on another processor. The concept of blocking delays on uniprocessors must now be extended to include delays caused by resource sharing with tasks on remote processors. For example, on any given processor P, a task can be preempted by a higher priority task on P, blocked by a lower priority task on P, and in addition wait for a task of any priority on remote processors to release a required global resource. Thus the determination of whether a task meets its deadline must consider not only preemption and blocking on its local processor but also the waiting time introduced by tasks of any priority on all remote processors. The multiprocessor priority ceiling protocol [29] is a synchronization protocol proposed to solve this problem in a multiprocessor system. The multiprocessor priority ceiling protocol generalizes the uniprocessor priority ceiling protocol by executing all critical regions associated with an unpreemptable shared resource on a particular processor called the synchronizing processor. As a result, the critical regions in a program are replaced by an invocation to a remote synchronization server. In this method, each task has a host processor on which most of the computation of that task takes place, and the task has a fixed priority on the host processor. The delay incurred by the task when executing on a synchronization processor is considered as the blocking time. Consequently the

worst-case response time for the task can be computed using existing methods [28]. Unfortunately a single centralized synchronization server will limit the scalability of the solution.

The major inadequacy in this approach is the inability to execute different tasks of an arbitrary transaction on different sites. All resources needed by all the tasks of a transaction may not be available on any one site. Allowing tasks of a transaction to execute at different sites provides the flexibility to deal with this situation.

Approach#2: This approach is based on independent scheduling on each processor. That is, tasks belonging to different transactions and allocated to different processors, would be scheduled on each processor independently such that the overall scheduling satisfies the end-to-end timing constraints.

This concept is proposed for establishing real-time channels in multi-hop networks [17]. Each link in the network is shared by different channels. Links and channels mimic processors and tasks respectively. Each link is considered as an independently scheduled unit. Schedulability analysis is based on the fixed priority scheduling analysis proposed in [1]. During the channel establishment phase, the priority assigned to the new channel at a link depends upon the characteristics of the other channels going through the link. The total response time, which is the sum of the response times over all the links on the route of the channel, is checked against the maximum permissible message delay and the channel can be established only if the latter is greater. How the permissible message delay is split among the different links is a major issue and could significantly affect the utilization of the links. In this work, worst-case phasing is assumed on each link. This assumption makes the analysis much simpler but can reduce the schedulability considerably.

In [44], instead of fixed priority scheduling, earliest deadline scheduling is proposed for establishing real-time channels. The earliest deadline scheduling is locally optimal and the zero phase assumption of fixed priority scheduling is relaxed. In [44], a heuristic for splitting the end-to-end deadline of real-time channels into deadlines for intermediate nodes is suggested.

In [3], fixed priority scheduling is used to schedule a set of periodic transactions in a flow shop. The priority of a transaction may change as it goes through processors in the flow shop. Some heuristics are proposed for assigning priorities to transactions on each processor. Each processor is considered as an independently scheduled unit and the worst-case phasing is assumed for analysing the system. In [3], a sufficient condition which guarantees the timing constraints for all transactions is presented. Sharing nonpreemptable resources is also considered in [3].

In this approach schedulability analysis should be done separately for each processor. To accomplish this, the traffic pattern for each processor should be known. Although the traffic pattern at the interface to the environment may be known, it does not imply a defined traffic pattern inside the system. One solution to this problem is making the traffic pattern predictable inside the system; i.e., the kernel on each processor inserts enough delay between two consecutive executions of each task. In [1], a different solution is proposed to overcome the unpredictability in the traffic pattern. It is assumed that all tasks arrive at the same time, but they might be blocked because of precedence relations. This blocking would cause a release jitter problem. The release jitter problem arises when a task is not released as soon as it arrives. With release jitter, a task may be released up to a bounded time J_i after it arrives. In [1], the effect of the released jitter is taken into account and an analysis method is suggested. All tasks are defined to arrive at the same time, but a precedence constrained task on one processor can have a release delay which depends on the

termination of all direct predecessors. The worst-case release jitter of such a task is computed by knowing the worst-case response times of direct predecessor tasks located on other processors. This approach reduces schedulability considerably. The computed release jitter for a task at the end of a transaction can be very large and consequently affects the response time of the transaction. In this approach there is no need to shape the traffic on each processor.

In [15], a different approach for meeting timing constraints is introduced. Although, the dynamic priority is used for scheduling tasks on each processor, the analysis of the real-time system is based on the *Producer/Consumer* paradigm. It is defined as a paradigm of process interaction wherein the i^{th} output of the producer must be consumed by the consumer before the $(i+1)^{st}$ output is produced. The idea behind this approach is to keep the rate of occurrence of events in the environment the same as the rate of consumption. The goal of scheduling at each node is to guarantee a constant flow of information on each channel; that is, the scheduling decisions at nodes do not consider end-to-end delays, which reduces their applicability.

Approach#3: Static Table Driven Paradigm

Static table-driven approaches are applicable to transactions that are periodic or have been transformed to periodic ones using known techniques [22]. In this approach, given transactions characteristics, a table is constructed for each processor using one of many possible techniques; e.g., using various search heuristics, that identify the start and completion time of each task. The schedule length is the LCM of the transaction periods.

The major characteristic of this paradigm is that it avoids blocking and it reduces the effect of interrupts. This paradigm can be used easily in multiprocessor and distributed systems and it can handle fault tolerance requirements, and resource requirements of transactions,

and precedence relationships among different activities of a transaction. In this approach, not only is a schedulability test undertaken but the actual schedule is constructed; i.e., the schedulability test is the construction of the run-time schedule. The construction of tables relies completely upon a priori knowledge of all the transactions behavior. Hence the scheme is workable only if all the transactions are effectively periodic, and the system is time driven. Generating the tables can be computationally expensive but high utilization of resources in the actual real-time system can be achieved.

The major drawback with this static approach is the lack of flexibility. This is manifest in the absence of any run-time policy. Minor changes can result in the need for complete redesign. This approach would fail if the behavior of the system depends on run-time decisions; e.g., an activity with an *if statement* where the result of the test causes the execution of another activity that requires different resources than the current activity.

Also, sporadic events are a source of problem in this approach. In this paradigm, sporadic events are changed to periodic ones based on the mapping proposed in [22]. If transaction $T_s(C_s, D_s, min_s)$ is a sporadic one with the given worst-case execution time (C_s), deadline (D_s), and minimum interarrival-time (min_s) then its equivalent periodic transaction satisfies the following conditions. Note that D_p is the deadline, Q_p is the period, and C_p is the execution time of corresponding periodic transaction; integer values are assumed for all parameters.

$$C_p = C_s; D_s \geq D_p; Q_p \leq \text{minimum}(D_s - D_p + 1, min_s)$$

For increasing the period of the periodic transaction to min_s , the deadline of the periodic transaction should not be close to the deadline of the sporadic one. That is, for sporadic transactions with a small slack-time this approach could result in the corresponding periodic transactions with a period much smaller than min_s . In fixed and dynamic priority par-

adigns this problem does not exist and the period of the corresponding periodic task is the same as the minimum interarrival time.

Another problem with the table driven approach, is the handling of tasks whose execution times are long compared to the period of highest rate cyclic task. This is a common situation in systems with multiple tasks in which a task with a long period requires significant processing to extract information, while another task requires a high sampling rate but with significantly less computation. In these cases the performance of this scheduling paradigm is very poor. The most common solution to such a problem is to arbitrarily break the long task into short ones. This practice, while it solves the immediate problem, is actually a process of preempting the long task by manually inserting the preemption points. This problem does not arise in the dynamic and fixed priority paradigms.

In this method, clock synchronization is unavoidable; i.e., the clocks of all processors must be synchronized. This could limit the range of applications to which this paradigm could be applied.

Another problem in this approach is the lack of consistency between the programming and run-time task representations; (i.e., the difference between tasks as defined by a programmer and the schedulable units at run-time). It means that the scheduler assumes it has task segments with known precedence and exclusion relations and resource requirements, while the programmer is writing code in terms of tasks. This is one reason why a nontrivial translation between the programming and run-time representation is required [19]. The non-trivial translation arises when the code for the tasks is compiled and a representation in terms of the task segments is extracted.

In [42], Xu extended the concept proposed in [41] to a multiprocessor system. He assumed a set of processes where each process consists of a sequence of segments. Each segment is

required to precede a given set of other segments and also excludes a given set of other segments. Each process has a release-time, a deadline and a computation-time. He has assumed M identical processors and proposed an algorithm to find a non-preemptive schedule (i.e., a segment cannot be preempted and preemption is allowed only at the boundary of segments) on processors such that each process starts executing after its release time and completes its computation before its deadline, and all the precedence and exclusion relations on segments are satisfied. Because of the non-preemptable segments, the algorithm is not optimal. This algorithm can also support both periodic and sporadic processes.

In [26], Ramamrithan has used the same approach but under a different model which makes it suitable for distributed systems. He assumed a set of periodic transactions such that each transaction consists of a set tasks with predefined resource requirements. Tasks may have precedence relationships and each transaction has its own predefined timing constraints. Given a set of periodic transactions, he introduced a heuristic algorithm that attempts to assign tasks of the transactions to processors in a distributed system and to construct a schedule of length L where L is the least common multiple of the transaction's periods.

Static scheduling in mutiprocessor or distributed systems is also investigated by the operations research community though under different title: *Job shop scheduling*. Given the schedule length L , the transaction's release times and deadlines, an explicit plan for the system can be generated using the existing job shop scheduling heuristics. As an example, in [31], a model of a job shop is presented which is similar to the model used in [26]. In the model it is assumed that there is a set of transactions and resources and the job shop scheduling requires scheduling the transactions on the resources. Each transaction consists of a set of tasks with precedence relationships and it has a release time and a deadline.

Each task has an execution time and in order to be successfully executed, it requires a set of resources. In [31], a backtrack search procedure is considered to generate a feasible schedule.

In [11], Fohler has considered the same model as in [26] but the assumption of transaction's deadline to be less than or equal to its period is relaxed. That is, a new instance of a periodic transaction can come into existence before the previous instance is being completed. In [11], it is shown that for these cases the schedule length L is greater than the least common multiple of the transaction periods and a heuristic algorithm for generating the schedule is presented.

2.3.2 Dynamic scheduling

Houng and Leung [14] have considered the problem of dynamic scheduling of independent tasks on a homogenous multiprocessor system. They demonstrated that no optimal policy exists when there are at least two distinct deadlines. They also showed that in the multiprocessor case, an optimal scheduler cannot be found unless a priori knowledge exists for the deadlines, computation times and arrival times of all the tasks.

In terms of sporadic tasks, which are one of the reasons for the overdesign of systems using static approaches, Stankovic et al [36], present a number of dynamic scheduling algorithms for multiprocessor systems. In their approach, all periodic tasks are statically allocated but sporadic tasks can migrate. Common to all of these algorithms is that they cannot guarantee the deadlines of all sporadic tasks and the scheduler dynamically builds explicit execution plans, rather than being priority driven.

In [6], Cheng et al have proposed a sophisticated algorithm for the dynamic scheduling of task groups in distributed hard real-time systems. A task group consists of tasks with precedence relationships and a hard deadline. The system is required to execute tasks in each

group completely or not at all. When a task group arrives at a node in the system, the system must decide whether tasks in the group can be executed before the deadline. In this approach scheduling of a task group needs the cooperation of all the nodes in the distributed system. To off-load scheduling overhead, it is assumed that each processor has a co-processor for executing system scheduling tasks. Because of cooperative scheduling, the hardware and software architecture of all nodes is almost the same. In this approach, regardless of the functionality expected from a node, each node requires a considerable amount of hardware and software complexity.

In [27] and [10] a method for integrating scheduling of static periodic transactions and dynamic arrivals is introduced. The basic idea behind the method presented is to find the earliest start times and latest start times for the statically scheduled periodic tasks, respecting the precedence constraints of the tasks. That is, instead of requiring that the tasks of the periodic transactions start at their scheduled start times, the schemes produce a scheduling window for each task. As long as a task is started within the window, the feasibility of all the tasks of a set of transactions will not be violated. Given a window for each statically allocated and scheduled task, more flexibility exists to decide if it is possible to delay the start of a task of a periodic transaction to run any aperiodic task that may have arrived. The first step in this method is to construct a static schedule for the periodic transactions. Next, the amount and distribution of unused resources, called spare capacities, is determined. Spare capacities are calculated off-line and incrementally maintained on-line.

2.4 Summary

Various scheduling approaches have been introduced for systems supporting either simple or complex real-time applications. A simple application is comprised of a single processing step with no precedence requirements. On the other hand, a complex application is comprised of multiple processing steps with precedence constraints and requiring end-to-

end timing checks. Timeline construction and algorithmic analysis are major approaches in scheduling and analyzing real-time systems.

In the timeline construction approach, the goal is to construct a timeline that statically binds the processing steps to specific time intervals on all resources. The timeline construction is time-consuming and any slight change to an application requires construction of a new timeline. This approach requires a global clock for the whole system.

In the algorithmic approach no explicit timeline is generated. Analysis is quick and changes in applications can be easily handled. Previous work using this approach has mostly focused on timing analysis for simple periodic applications executing on single resource. Some recent work has explored end-to-end timing analysis for complex applications [3, 15, 17, 42]. In all these works the multi-resource scheduling problem is decomposed into a set of single resource scheduling problems and each resource can use its own independent scheduling method. In this approach no global clock is required. The challenges in this approach are allocating tasks to processors and mapping end-to-end timing requirements to intermediate tasks timing requirements. Using fixed priority scheduling on resources, the key issue in the algorithmic approach is *priority assignment* (i.e., assigning priorities to transactions tasks such that transactions end-to-end deadlines are met); in [3], it is shown that this problem is NP hard. It should be emphasized that the use of priorities is not the source of complexity. Timeline models also have to solve complex graph reductions problems in order to find a feasible timeline for the complete system.

Chapter 3

The System Model and Scheduling Method

3.1 System Model

A real-time system is represented as a set of processors such that each processor has its own local memory and runs its own kernel. Each processor runs a set of tasks and the behaviors of the tasks are predefined. Physically, we do not consider how processors are wired together, but because of the concept of transaction, logically, we considered a wired architecture for the system. At the functional level, the whole system is event-driven and each event can trigger a transaction through the system. A transaction is processed by different tasks on different processors until terminated somewhere in the system. In this model, each task is assigned to a specific transaction and no two transactions use the same task. In the case of sharing a task by several transactions, we consider several instances of the task and assign each instance to a different transaction. With this assumption the priority of a transaction is the same as the priorities of tasks that it traverses. The priority of a transaction may change as it is going through different processors. It is assumed that a transaction does not change its priority while in a processor. Assigning tasks to different processors is not arbitrary and tasks are assigned to specific processors.

Figure 3.1 shows an example illustrating the model of real-time systems used in this research. Processor#4 handles a non-preemptable shared resource and transactions may share the resource when going through this processor. Also, it is assumed that a transaction may pass through a processor more than a once, as transaction t_3 has visited processor#2 twice. A transaction may also interact with its own sub-transactions. In Figure 3.1

transaction t_1 forks sub-transaction t_2 at point A and then waits for it at point B. In other words the continuation of transaction t_1 required the termination of subtransaction t_2 . In this model transactions interact with each other only through shared resources.

Figure 3.1 The System Model

It is assumed that a transaction does not change its priority while it is passing through a processor. A transaction may pass through different tasks while it is in a processor, but all the tasks have the same priority. Figure 3.2 shows transactions while they are passing through a processor. In Figure 3.2, it is assumed that tasks $T1$, $T2$ and $T3$ have the same

priority. Also tasks $T4$ and $T5$ are assigned the same priority but not necessarily the same as the set of tasks $T1$, $T2$ and $T3$.

Figure 3.2 Transactions Passing Through a Processor

3.2 Scheduling Method and Schedulability Criteria

In a m -processor model, without revisiting, each transaction t_i may see at most m tasks and spend a worst-case time of R_{ij} on processor j ; note: multiple tasks for a transaction in a processor are lumped together and scheduled as a single task. Each transaction t_i has a period Q_i and when it is necessary to distinguish individual tasks corresponding to a transaction and belonging to different periods, the symbol $T_{ij}(k)$ is used. $T_{ij}(k)$ denotes the k^{th} activation of transaction t_i when going through the processor j . It is assumed that each transaction follows a predefined route through the system and hence a task $T_{ij}(k)$ cannot begin until the task $T_{i(j-1)}(k)^1$ is completed. This dependency between tasks makes the

1. The symbol $(j-1)$ refers to the processor that transaction i has passed through before processor j . If j is the first processor the transaction goes through, then $j-1$ refers to the last processor visited by the transaction.

scheduling of transactions difficult, but is simplified by scheduling each processor independently of the other processors in the system. This idea is similar to the one proposed by Betatti for a periodic flow-shop [3] and Kandlur for establishing real-time channels [17]. Before going to the details of the method, some theorems on fixed priority preemptive scheduling are reviewed.

Theorem 3.1 [21]:

Under fixed priority preemptive scheduling, regardless of how priorities are assigned to tasks, the worst-case response time for each task occurs when all tasks are initiated at the same instant. This is true for both periodic and sporadic tasks as long as the minimum interarrival time of a sporadic task is considered to be its period.

Theorem 3.2 [1]:

Assume a set of n independent tasks $\{T_1, \dots, T_n\}$, with worst-case execution time, $\{C_1, \dots, C_n\}$ and periods $\{Q_1, \dots, Q_n\}$. There are n different priority levels that could be assigned to the tasks arbitrarily. For a specific priority assignment, the worst-case execution time for an arbitrary task l is calculated by solving the following equation:

$$R_l = C_l + \sum_{\forall k | (k \in hp(l))} \left\lceil \frac{R_l}{Q_k} \right\rceil C_k \quad (\text{EQ 3.1})$$

where R_l is the worst-case response time of task T_l and $hp(l)$ is the set of all tasks of higher priority than task l .

In [1], an iterative method for calculating R_l is proposed. In this method, each time a value for R_l is calculated, its validity is checked. The algorithm checks, in sequence, the possible values for R_l until it finds the right one. The iterative method is described in Equation 3.2.

$$R_l^{n+1} = C_l + \sum_{\forall k \in hp(l)} \left\lceil \frac{R_l^n}{Q_k} \right\rceil C_k \quad (\text{EQ 3.2})$$

The initial value for R_l is C_l . The above equation is guaranteed to converge if the processor utilization is less than or equal to 1 [1]. The above analysis makes the assumption that the worst-case response time of task l must be less than its period.

Using the algorithm, the worst-case response time can be calculated for each task residing on the processor j . In other words, it can be guaranteed that every task $T_{ij}(k)$ is completed by R_{ij} units of time after it becomes ready. To compute the worst-case response of each task on each processor, the traffic pattern entering each processor must be periodic. Although the traffic at the boundary of the real-time system is periodic, there is no guarantee that this traffic remains periodic inside the system. For solving this problem, the traffic entering each processor inside the system must be reshaped to periodic. This can be done by delaying the ready-time of task $T_{ij}(k+1)$ to the ready-time of task $T_{ij}(k)$ plus the period of the task. An arbitrary transaction t_i can meet its deadline if the following inequality is satisfied.

$$\sum_{\forall j | j \in \{t_i\}} R_{ij} \leq D_i \quad (\text{EQ 3.3})$$

where D_i is the deadline of transaction t_i and $\{t_i\}$ is the sequence of processors that are traversed by transaction t_i .

In this model, The worst-case response time of a transaction when passing through an arbitrary processor depends only on the relative priority of the transaction on that processor. By changing the priorities of transactions during their travel through the system, it may be possible to find a proper schedule such that all transactions satisfy their timing

constraints. Now the question is how to assign priorities to transactions for a given system such that timing constraints are satisfied. This issue is considered in later chapters.

So far, only independent transactions have been considered. Now, the case that transactions share a physical or logical resource is discussed. It is assumed that for sharing a specific object, transactions should go through a specific processor. The goal is to guarantee that transaction t_i when passing through the processor does not spend more than an upper-bound on the processor. Before extending the schedulability criteria, a theorem based on the priority inheritance synchronization protocol is introduced.

Theorem 3.3 [1]:

A set of tasks $\{T_1, \dots, T_n\}$, running on a processor, share resources and have to go through critical sections. Using the priority inheritance synchronization protocol, the worst-case response time of task l is found by solving the following equation

$$R_l^b = C_l + B_l + \sum_{\forall (k \in hp(l))} \left\lceil \frac{R_l^b}{Q_k} \right\rceil C_k \quad (\text{EQ 3.4})$$

where B_l is the worst-case blocking time that is seen by task l and R_l^b is the worst-case response time of the task when blocking is taken into account. This equation can also be solved recursively.

Although the worst-case response time cannot be computed in polynomial time, an upper bound can be calculated with polynomial time complexity. With some approximation (i.e., by equating $\left\lceil \frac{R_l^b}{Q_k} \right\rceil = \left(\frac{R_l^b}{Q_k} + 1 \right)$) the above equation can be written as:

$$R_l^b \leq R_l^{b+} = C_l + B_l + \sum_{\forall k \in hp(l)} \left(\frac{R_l^{b+}}{Q_k} + 1 \right) C_k \quad (\text{EQ 3.5})$$

In the worst-case, this approximation adds $\sum_{\forall k \in hp(j)} C_k$ to the response time of task l .

Based on the above approximation, and by doing some mathematical manipulation the upper-bound for worst-case response time is given by:

$$R_l^b = \frac{\left(B_l + \sum_{k=1}^j C_k \right)}{1 - \sum_{k=1}^{j-1} \frac{C_k}{Q_k}} \quad (\text{EQ 3.6})$$

In the above equation, it is assumed that priority of tasks decreases as their index increases.

Now, using theorem 3.3, the scheduling criteria can be extended as follows. For each transaction t_i , the following inequality should be satisfied. In this inequality R_{ij}^b refers to the worst-case response time of transaction t_i on processor j when the effect of blocking is also taken into account.

$$\left(\sum_{\forall j | (j \in \{t_i\})} R_{ij}^b \right) \leq D_i \quad (\text{EQ 3.7})$$

3.3 Applying the Scheduling Method to more Complex Domains

The next assumption made in the model is revisiting; that is, a transaction that passes through a processor more than once. Figure 3.3 shows this case; processor $P3$ is visited twice by transaction $t1$.

To find the worst-case response time of transaction t_1 , it is sufficient to follow the transaction and calculate its response time each time it passes through a processor. For transaction $t1$, the response time is the sum of response time for tasks $T1$ to $T7$.

Figure 3.3 Revisiting in the Model

For calculating the response time of task $T1$, the effect of tasks $T2$ and $T3$ should not be taken into account, because they belong to the same transaction and pipelining is not assumed. This fact is considered for computing the response time of all the tasks belonging to the transaction t_1 . The effect of revisiting is considered for other transactions also. In Figure 3.3 transaction t_2 is affected by transaction t_1 in a complex way. If tasks $T7$ and $T8$ are released at same time and both tasks $T7$ and $T6$ have higher priority than $T8$ then task $T8$ may also be affected by task $T6$ depends on the time transaction t_1 spend in processor $P1$ by going through task $T2$. One approach to deal with this problem is assuming that $T8$ is affected always by both $T6$ and $T7$ and consider these two tasks as one task with execution time equal to sum of execution time of task $T6$ and $T7$. This approach can be very pessimistic; to be more accurate, the worst case response time of task $T8$ should be calculated two times: (1) when $T8$ is released at the same time as $T6$ and, (2) when $T8$ is released at the same time as $T7$. The larger response time should be considered as the final

worst-case response time of task T_8 . Figure 3.4 shows this approach. In the case, when task T_8 is released at the same time as T_7 , task T_8 is finished at point A; when it is released at the same time as T_6 it is finished at point B. So, point A should be considered as the worst-case execution time of transaction t_2 . In this example, it is assumed that the response time of T_8 is affected only by one activation of transaction t_1 . The activation time of task T_6 depends on the response time of tasks T_7 and T_2 . By assuming zero phasing for T_7 , the activation time of T_6 could not be earlier than sum of the execution time of the tasks T_7 and T_2 .

Figure 3.4 Computing the Worst-Case Response Time of a Task in the Face of a Revisiting Transaction

For determining the effect of a revisiting transaction t_i on other transactions passing through an arbitrary processor P_j , it is necessary to determine how different tasks of transaction t_i , running on processor P_j , may affect other transactions passing through this processor. Two different scenarios may happen. In Figure 3.5, assume that priority of T_1 , T_2 , and T_4 is higher than T_3 . In this case, by assuming the zero phasing condition for each

task $T1$, $T2$, and $T4$ separately, the response times of task $T3$ are calculated. The three response times are compared and the highest is selected as the worst-case response time of task $T3$. When the zero phase assumption is made for one of the tasks $T1$, $T2$, and $T4$, the phasing of other tasks must account for precedence relations among tasks. For example, if zero phase is assumed for task $T1$, the phase of $T2$ would be equal to the amount of time required for transaction t_i to reach to task $T2$ which is equal to response time of task $T1$ plus the time transaction t_i spends in processor P_k . The phase for $T4$ is equal to response time of tasks $T1$ and $T2$ plus the time transaction t_i spend on processor P_l . In this scenario, subsequent invocations of transaction t_i may also affect transaction t_j .

Figure 3.5 Different Ways a Revisiting Transaction Affects the Worst-Case Response Time of other Tasks

The next scenario corresponds to the case that at least one of the tasks $T1$, $T2$, or $T4$ has a lower priority compared to $T3$. If task $T2$, for example, has a lower priority compared to $T3$, then only combinations $(T1)$ and $(T4, T1)$ should be considered for determining the worst-case effect of transaction t_i on transaction t_j . The reason is that task $T3$ always finishes before task $T2$ and at most two invocations of transaction t_i may affect transaction t_j .

In Figure 3.5, the combination (T_4, T_1) has the worst-case effect because task T_4 is the last task of the transaction t_i and the next invocation of transaction t_i can start immediately after the current invocation of the transaction is terminated.

Now an algorithm is introduced to compute the worst-case response time of an arbitrary task T_i belonging to a transaction and running on a given processor in the case that revisiting transactions exist in the system. The algorithm is based on the following iterative equation. This equation determines the worst-case response time when revisiting transactions do not exist in the system. In this equation, C_i is the execution time and Q_i is a period of the task. This equation, on each iteration, determines the next possible response time of the task T_i .

$$R_i^{n+1} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^n}{Q_k} \right\rceil \times C_k \quad (\text{EQ 3.8})$$

The above equation is extended to consider revisiting transactions. In Equation 3.9, $hptask(T_i)$ shows all the tasks with priority higher than task T_i such that none of them belong to any revisiting transactions and running on the same processor as T_i . $hptrans(T_i)$ corresponds to all the revisiting transactions, except the one to which T_i belongs, with at least one task with priority higher than task T_i . Note that other tasks belonging to the same transaction as T_i , do not contribute to the worst-case response time of task T_i . Let $load(t_z, T_i, \tau)$ represents the maximum CPU time requested by transaction t_z during time window $[0, \tau)$ which contributes to the response time of the task T_i , then the worst-case response time of task T_i (i.e., R_i) is given by solving the following equation:

$$R_i = C_i + \sum_{(\forall T_k) \in hptask(T_i)} \left\lceil \frac{R_i}{Q_k} \right\rceil C_k + \sum_{(\forall t_k) \in hptrans(T_i)} load(t_k, T_i, R_i) \quad (\text{EQ 3.9})$$

Equation 3.9 can be solved using iterative Equation 3.10.

$$R_i^{n+1} = C_i + \sum_{(\forall T_k) \in hptask(T_i)} \left\lceil \frac{R_i^n}{Q_k} \right\rceil C_k + \sum_{(\forall t_k) \in hptrans(T_i)} load(t_k, T_i, R_i^n) \quad (\text{EQ 3.10})$$

Equation 3.10 starts with a window of size 0 for the task's response time (i.e., $R_i^0 = 0$). In each iteration of the equation the amount of demand for CPU in the current window is calculated and the window size is increased to the amount of CPU demand. The iterations continues until the current window size and the CPU demand for the current window become equal. The amount of demand for CPU in the current window is calculated by summation of the following three terms: (1) the task execution time (C_i) which is fixed for every window, (2) the amount of CPU time needed in the current window by the tasks belonging to the set $hptask(T_i)$ and, (3) the amount of CPU time needed in the current window by tasks with priority higher than T_i and belonging to revisiting transactions in the set $hptrans(T_i)$. Note that the above three terms corresponds to the terms in Equations 3.10.

The algorithm for computing the $load()$ is described in the Table 3.2. In this algorithm, $T_{z,n,j}$ refers to the task that belongs to a revisiting transaction t_z and is the n^{th} revisit of the transaction after task $T_{z,0,j}$ on processor j . Assume that, there are r revisits after $T_{z,0,j}$ then $T_{z,r,j}$ corresponds to the task representing the last revisit and $T_{z,r+1,j}$ refers to the task $T_{z,0,j}$ but corresponds to the next invocation of the transaction. The algorithm for computing the $release()$ is given in Table 3.1. In this algorithm Q_i is the period of the transaction t_z and N is the number of revisits of this transaction on processor j . In this algorithm PR refers to the execution time of a task and $NEXT^k(T_{z,r,j})$ refers to the k^{th} task following task $T_{z,r,j}$ in transaction t_z . As an example in Figure 3.5, $NEXT^0(T1) = T1$ and $NEXT^1(T1)$ is $T5$. In this figure, $NEXT^5(T1)$ refers to the next invocation of task $T1$. PRI in the algorithm $load()$ refers to the priority of a task and the symbol $C_{z,r,j}$ shows the execution time of task $T_{z,r,j}$.

In the algorithm *release()*, it is assumed that response time of a task is equal to its execution time. This assumption is pessimistic because response time is usually greater than execution time because of higher priority tasks. This assumption may result in higher upper-bounds for the end-to-end delay of the transactions.

RELEASE($T_{z,r+n,j}$)

$$\text{release} = \left\lfloor \frac{n}{N} \right\rfloor \times Q_i; \quad m = n - N \times \left\lfloor \frac{n}{N} \right\rfloor; \quad k = 0$$

- 1) if ($T_{z,r+m,j} = \text{NEXT}^k(T_{z,r,j})$) then return(release)
- 2) release = release + PR ($\text{NEXT}^k(T_{z,r+n,j})$)
- 3) $k = k + 1$; go to 1

Table 3.1 Algorithm RELEASE

LOAD(t_z, T_i, τ):

- 1) determine all tasks belonging to t_z on processor j with priority higher than T_i . This set is represented by $hp(t_z, T_i)$; initialize max_load to zero
- 2) select next task from the set $hp(t_z, T_i)$; if there are no more tasks return(max_load)
- 3) assume zero phasing for the task selected in (2); assume this task is $T_{z,r,j}$
- 4) initialize load to zero; initialize n to zero
- 5) load = load + $C_{z,r+n,j}$
- 6) if load $< \tau$ then increment n else go to 8
- 7) if ($\text{PRI}(T_{z,r+n,j}) > \text{PRI}(T_i)$ and $\text{RELEASE}(T_{z,r+n,j}) < \tau$) then go to 5 else continue
- 8) if load $> \text{max_load}$ then max_load = load
- 9) go to 2

Table 3.2 Algorithm LOAD

As an example of the worst-case response time algorithm for revisiting, consider Figure 3.6 in which 3 transactions pass through processors $P1$ and $P2$. The work load characteristics of the transactions are given in Table 3.3. It is assumed that $T1$ has the highest priority and $T4$ has the lowest. We are not concerned about the priority of the tasks $T5$ and $T6$ on processor $P2$. Transactions t_2 is a revisiting transaction and we are interested in the worst-case response time of transaction t_3 . The worst-case response time of transaction t_3 is

equal to worst-case response time of task T_4 . For task T_4 , $hptask(T_4)$ is $\{T_1\}$, $hptrans(T_4)$ is $\{t_2\}$ and $hptask(t_2, T_4)$ is $\{T_2, T_3\}$.

Figure 3.6 An Example for Computing Worst-Case Response Time in the case of Revisiting Transactions

	Execution Time on Processor#1	Execution Time on Processor#2	Period of The Transaction	Deadline of The transaction
1	$C1=2$	0	$Q1=4$	$D1=4$
2	$C3=3$ $C2=2$	$C5=10$ $C6=10$	$Q2=40$	$D2=40$
3	$C4=2$	0	$Q3=20$	$D3=20$

Table 3.3 Workload for the Example of Figure 3.6

In the first iteration of the algorithm, the response time is 2 which is the execution time of task T_4 . In the second iteration, for the period $(0,2)$ transactions t_1 and t_2 generate a maximum load of 2 and 3, respectively. In next iteration, in the period $(0,7)$ the load generated by transaction t_1 is 4 and by transaction t_2 is 3. In the last iteration, for the period $(0,9)$ the load generated by t_1 is 6 and by t_2 is still 3. The algorithm converges on the fourth iteration and the worst-case response time for transaction t_3 is 11.

$$R_4^1 = C_4 = 2$$

$$R_4^2 = C_4 + \left\lceil \frac{R_4^1}{Q_1} \right\rceil \times C_1 + \text{load}(t_2, T_4, R_4^1) = 2 + 2 + 3 = 7$$

$$R_4^3 = C_4 + \left\lceil \frac{R_4^2}{Q_1} \right\rceil \times C_1 + \text{load}(t_2, T_4, R_4^2) = 2 + 4 + 3 = 9$$

$$R_4^4 = C_4 + \left\lceil \frac{R_4^3}{Q_1} \right\rceil \times C_1 + \text{load}(t_2, T_4, R_4^3) = 2 + 6 + 3 = 11$$

To show how the algorithm *load()* works, we look in more detail at the last iteration of the above example. The amount of worst-case load imposed by transaction *t2* on task *T4* in time interval (0,11) is required. The zero phase is checked for the tasks *T2* and *T3* separately to determine which case imposes the worst-case load. In the case that task *T3* has a zero phase, the release of task *T2* is at time $C3+C5=13$ which does not contribute to the load, cause it is out of interval (0,11). So in the case that the zero phase is assumed for *T3*, the worst-case load imposed by transaction *t2* on task *t4* in the time interval (0,11) would be equal to $C3=3$. By considering zero phase for the task *T2*, the release of task *T3* happens at time $C2+C6=13$. In this case, the release of *T3* does not contribute to the load and the worst-case load is $C2=2$. When *T3* has the zero phase the load is higher, thus this case is selected as the worst scenario, and the load for the interval (0,11) is 3.

Another feature in the model is a series of transactions, each one triggered by the previous one and each one has its own period and timing requirement; this case is shown in Figure 3.7.

Figure 3.7 Series of Transactions

In Figure 3.7 transactions t_1 and t_2 are in series: transaction t_2 is triggered by transaction t_1 and each transaction has its own timing requirement and period. Transaction t_2 may be triggered once for each N activations of t_1 . In this case the period of t_2 would be $1/N$ of the period of t_1 . Here, when computing the worst-case computation time of task $T2$ the effect of task $T1$ should be taken into account (if priority of $T1$ is higher than $T2$). The effect of $T1$ must be taken into account since during the execution of $T2$, task $T1$ can also become activated.

The last feature of the model is the possibility of creating sub-transactions. Figure 3.8 shows this feature.

Figure 3.8 Interaction of a Transaction with its Subtransaction

To compute the end-to-end latency of transaction t_1 , it is necessary to determine which transaction, t_1 or t_2 has the longest latency between points A and B; the longest one should be considered as the one that contributes to the worst-case latency for transaction t_1 . To find the upper-bound on the end-to-end delay of the transaction t_1 , first, R_1 , R_2 , R_3 and R_4 , the worst-case response time of tasks $T1$, $T2$, $T3$ and $T4$ are computed; then, the worst end-to-end delay is equal to $R_1 + \max(R_2, R_4) + R_3$. The worst-case response time corresponds to the critical path [38].

3.4 Discussions

A model for a class of hard real-time systems has been proposed and a scheduling method was presented. The model supports real-time systems that consist of a set of transactions. A transaction is composed of a set of precedence constraint tasks, where each task is running on a different processor. Transactions are triggered periodically and they have end-to-end timing constraints (i.e., a constraint on end-to-end response time). In the model, a

transaction can visit a processor several times and it can fork parallel subtransactions. The scheduling method introduced in this chapter decomposes multiprocessor system timing analysis into a set of single resource timing analyses that can predict the performance of system and can be used to determine that all transaction timing requirements are met. Some assumptions are made that restrict the usefulness of the model for more general cases; these assumptions are listed below. Also, the scheduling issues are discussed.

ASSUMPTIONS:

- *It is assumed that independent transactions do not interact with each other (other than indirectly through shared resources). Two transactions are independent if the triggering of one does not depend on the other transaction.*

In some real-time applications, transactions may actually interact with each other. For example in a multi-loop control system, multiple sensors may be used to design an optimal controller. Each sensor may initiate a separate transaction and data from different sensors may be used by the controller task. Data from different sensors may be written in a shared data structure on the controller processor and used by the controller task. In this example, transactions interact on the controller processor through a shared data structure. It may be possible to model applications like this by assuming a worst-case execution time for tasks that are dependent on other transactions; however, this may reduce the ability of algorithms to find a feasible priority assignment.

- *It is assumed that a transaction does not change its priority while it is passing through a processor.*

The tasks belonging to a transaction may have precedence constraints. Note that for a flow shop model the precedence constraints are strictly linear. In this chapter, it is assumed that all the precedence related tasks running on the same processor and belonging to the same

transactions have the same priority. Relaxing this assumption has some benefits: (1) if tasks are allowed to change their priorities then higher schedulability can be achieved and processors can be better utilized, (2) for some applications, we need to change priorities. Assume a task that goes through different phases of reading a sensor, doing a computation, and sending the result to the next processor. If the application requires that reading sensor action has a low jitter then this action must be assigned to a high priority. However, the computation action may require lots of CPU time and assigning it to a high priority (same as sensor reading action) may degrade substantially the schedulability of other tasks allocated to the same processor.

A potential solution to this problem is to adjust the phase of all precedence related tasks such that they are initiated periodically and treat them as periodic tasks. This method would require further research.

- *The allocation problem is outside the scope of the model and it is assumed that the route for each transaction is predefined.*

The allocation problem is separated from the priority assignment problem. This is not unrealistic since tasks must often be allocated to specific processors for reasons other than meeting deadlines; e.g., access to I/O devices, access to special processor types, and memory resource balancing. Regardless, the allocation of task to processors should, as much as possible, be done so as to extend the space of feasible solutions, thus increasing the likelihood that a heuristic priority assignment algorithm can find a solution.

- *It is assumed that each transaction follows a predefined route for all of its invocations and changes of route caused by branches are not considered.*

In some real-time applications a transaction may change its path during run-time following the detection of a condition. For these cases, one solution is to determine all possible

scenarios at design time and analyse each scenario separately such that for each scenario, all transactions meet their end-to-end deadlines. This approach may however result in an overdesign of the system.

- *For accessing a given shared resource, it is necessary that the transactions go through a specific processor.*

If a processor is responsible for a shared resource then transactions that require access to the resource must visit this processor. However, sometimes the shared resource is not managed by a specific processor. As an example, different tasks may send each other messages through a shared communication link. In these cases, the shared resource could be modeled as a processor and resource access as a task on the processor, or the worst-case response time of access could be added into the execution time of the task that requires access to the resource. In the latter case, the model is simpler, but more pessimistic.

- *It is assumed that task execution time is known.*

Task execution times might not be easy to predict (due to instruction pipelining, etc.). This assumption must be verified in actual system.

SCHEDULING METHOD ISSUES:

- *The scheduling algorithms are not optimal; in the sense that priority assignment algorithms may not be able to find a schedule when one may exist.*

If no feasible solution is found by priority assignment algorithms then problem may be solved by reallocating some of the tasks among the processors. This may increase the feasibility space and make it easier for heuristic algorithms to find a solution.

- *On each processor, zero phasing is assumed to compute the response-time on the processor.*

The zero phasing assumption is pessimistic, especially when transactions go through several processors; but it makes the analysis less complicated. This may reduce the likelihood of detecting a solution.

Chapter 4

Heuristics For Priority Assignment in the Flow-Shop Model

4.1 Introduction

In this chapter, the problem of assigning priorities to different transactions tasks running on different processors is investigated. This problem is proved to be *NP-hard* in [3]. In [3], a few heuristics are introduced for priority assignment but the performance of the heuristics is not evaluated.

In this chapter, the performance of a number of heuristics are analyzed. Both existing heuristics as well as heuristics introduced by this research are considered. The performance criteria is the ability of the heuristic to find a feasible solution. This is expressed as a proportion of the test cases for which a solution is found.

In general, the performance of heuristics can be analyzed in several ways, such as: empirical analysis; worst case analysis; and probabilistic analysis. Empirical analysis is based on programming the heuristic and trying it on a random sample of problems. Worst-case study establishes the maximum deviation from optimality when the heuristic is applied within a given problem class. In a probabilistic approach, a density function for the problem data is assumed and the probabilistic properties of a heuristic are determined. The probabilistic properties may be a bound on the probability that heuristic finds a solution within a prespecified percentage of optimality. The method used here is based on empirical analysis. Empirical analysis is chosen because of the following reasons.

- It does not rely on proving hard worst-case and average-case analysis theorems. Furthermore, heuristics may have overlapping coverage of the feasible solutions and execute quickly, so that there is no benefit in proving that one algorithm is better than another. As the results will show later, none of the heuristics completely subsumes the solutions of another heuristic.
- Unlike probabilistic analysis, it need not restrict itself to a simple and unrealistic distributions of random variables.
- Unlike worst-case analysis, it can be used to compare different heuristics.
- Empirical methods can help to determine how a heuristic performs on a typical class of problems, including how performance of the heuristic depends on problem characteristics. Empirical analysis can also help to identify weaknesses of a heuristic.

Checking for the existence of a feasible schedule is not tractable in general. For this reason, we do not consider the *success rate* (the probability of the heuristic being successful in generating a feasible schedule, given that such a schedule exist) as a measure of the performance of the heuristics; instead, we compare the performance of different heuristics with respect to each other.

4.2 Different Priority Assignment Strategies

In this section, besides describing existing heuristic algorithms (LM and HOPA) for the priority assignment problem, three new heuristic algorithms (EQF, RTC and WP) are introduced. We will show later, through simulation, that the new heuristics improve the coverage of existing ones (i.e., they are able to find feasible solutions in many cases that the existing heuristics cannot). In this section, the algorithms are described and the motivation behind the new heuristics are given.

In the LM and EQF approaches, priority assignment is done on a per processor basis; i.e., a processor is selected and priorities are assigned to all tasks on that processor. After computing the worst-case response time for all the tasks, another processor is selected for priority assignment. In the RTC approach, priority assignment is done on a per transaction basis. In this method, a transaction is selected and the slack time for each of its tasks is determined. This process is repeated for all transactions. In the WP method, in each step of the algorithm, a priority is assigned to one of the tasks. In each iteration the current worst position (an unassigned priority level with highest response time) is determined and an unscheduled task is selected for that position. The unscheduled task is chosen such that its corresponding transaction has the highest flexibility after assigning to the worst position. The algorithm terminates when all the tasks are scheduled. The HOPA algorithm is based on the *redistribution* of the end-to-end deadlines of each transaction among its tasks. At each iteration, using assigned local deadlines, the worst-case response time for each task is computed. The worst-case response times and the current local deadlines are used to compute the new local deadlines. The algorithm terminates if no feasible solution is found after a given number of iterations.

In the LM, EQF, RTC and HOPA algorithms, the problem of priority assignment is related to the problem of how end-to-end deadlines are mapped to local deadlines (task deadlines). These two problems are not necessarily the same. After assigning deadlines to tasks, different priority assignment strategies on a local processor can be used to assign priorities. The priority assignment strategy can consider the assigned deadlines as one of the factors for making the final decision on how to assign priorities on a local processor. In all the above algorithms deadline monotonic scheduling is used for assigning the priorities to tasks on a processor. This means, a higher priority is assigned to a task with the lower deadline. With this in mind, the problem of priority assignment becomes the deadline assignment problem. In WP, however, deadlines are not assigned to tasks of transactions. In this method, in each step of the algorithm, the priority of one of the tasks is determined.

In a m -processor model each arbitrary transaction t_i consists of m tasks $T_{i1}, \dots, T_{im}$ to be executed in series, each one on a different processor. In other words, each transaction goes through m different stages, each stage corresponding to a processor. If a transaction does not visit a specific processor a zero execution time can be assumed for the task of the transaction on that processor. A deadline assignment strategy determines the values of $dl(T_{ij})$ for each T_{ij} ($1 \leq j \leq m$) where $dl(T_{ij})$ is the local deadline of the task T_{ij} . The five different deadline assignment strategies are introduced below.

(1) Laxity Monotonic (LM)

In this strategy, the local deadline of each task at stage j is computed as follows:

$$dl(T_{ij}) = dl(t_i)|_j - \sum_{k=j+1}^m C_{ik} \quad (\text{EQ 4.1})$$

In the above equation, $dl(t_i)|_j$ refers to the deadline of transaction t_i at stage j and C_{ik} refers to the worst-case processing time of task T_{ik} . The deadline of a transaction at stage j is computed as follows:

$$dl(t_i)|_j = dl(t_i)|_{j-1} - R_{i(j-1)} \quad (\text{EQ 4.2})$$

In this equation $j > 2$ and $R_{i(j-1)}$ refers to the worst-case response time of task $T_{i(j-1)}$. Note that at stage 1 the deadline of the transaction is the same as its end-to-end deadline.

Using equation 4.1, at each stage, local deadlines are computed for the tasks of transactions and priorities are assigned to the tasks based on deadline monotonic scheduling. Worst-case response times are calculated for all the tasks, as described in theorems 3.2 and 3.3 in Chapter 3, and the new transaction deadlines are computed by subtracting the worst-case response time of each task from its corresponding transaction deadline. This method is suggested in [3] for assigning priorities to transactions on each processor.

Transaction Number	Execution Time on Processor #1	Execution Time on Processor #2	Deadline of The Transaction	Period of The Transaction
1	1	2	4	4
2	3	2	15	15
3	0.25	1	5	5
4	2	1	20	20

Table 4.1 A Transaction Set

Table 4.2 shows how this heuristic algorithm works for the transaction set given in Table 4.1. The transaction set consists of four transactions going through two processors. Each transaction is characterized by a four-tuple: (1) the processing time on the first processor; (2) the processing time on the second processor; (3) the deadline of the transaction; and (4) the period of the transaction.

The deadlines assigned on the first processor are the effective deadlines of the transactions; i.e., the end-to-end deadline of each transaction minus the execution time of the transaction on the second processor. Using deadline monotonic scheduling for assigning priorities to tasks, the worst-case response time of each transaction on the first processor is computed. The third column in Table 4.2 corresponds to these worst-case response times.

transaction	deadlines on the first component	worst-case response time	deadlines on the second component	worst-case response time
1	2	1	3	2
2	13	5.5	9.5	8
3	4	1.25	3.75	3
4	19	7.5	12.5	12

Table 4.2 Deadline Assignment Using LM Algorithm

Subtracting the worst-case response times from the end-to-end deadlines of transactions, the new deadlines of transactions are computed. For the second processor, the new deadlines are the effective deadlines. Again, using deadline monotonic scheduling for assigning priority, the worst-case response times of the transactions on the second processor are computed. These worst-case response times should be less than the assigned deadlines to make the priority assignment a feasible schedule, which is the case for the transaction set introduced in Table 4.1.

(2) Equal Flexibility (EQF)

In this method, the local deadline of each task at stage j is computed as follows:

$$dl(T_{ij}) = C_{ij} + \left(dl(t_i) \Big|_j - \sum_{k=j}^m C_{ik} \right) \times \left(\frac{C_{ij}}{\sum_{k=j}^m C_{ik}} \right) \quad (\text{EQ 4.3})$$

The first bracket is the remaining slack time at stage j for the transaction t_i and the second one is the portion of slack time that should be assigned to task T_{ij} . This algorithm is introduced as a part of this research to improve the performance of LM by considering the future work of a transaction for assigning deadlines to its tasks at each stage. By not assigning all the remaining slack to tasks at each stage, we expect to improve the performance of the algorithm.

Table 4.3 shows how this heuristic algorithms works for the transaction set given in the Table 4.1. The assigned deadlines on the first processor are not the effective deadlines of the transactions. The deadlines are assigned based on the EQF formula. Deadline monotonic scheduling is used for assigning priorities to the tasks.

transaction	deadlines on the first component	worst-case response time	deadlines on the second component	worst-case response time
1	1.33	1.25	2.75	2
2	9.6	5.5	9.5	8
3	1	0.25	4.75	3
4	10	7.5	12.5	12

Table 4.3 Deadline Assignment Using EQF Algorithm

(3) Real-time Channel (RTC)

In the last two methods, the decision for priority assignment is made on a per processor basis. Another way is to assign priorities based on transactions. In this method, after selecting a transaction, its lowest possible response time on each processor is determined and the total remaining slack time (i.e., the end-to-end deadline of the transaction minus the summation of minimum response times) is divided between the transaction tasks based on a specific strategy. The computed slack time for each task is added to its minimum response time to find the local deadline of the task. It is necessary to ensure that the new transaction does not affect the guaranteed delivery times of existing transactions. This is guaranteed by meeting the local deadlines of scheduled tasks on each processor when scheduling a new transaction. Using fixed priority scheduling the following algorithm can be used to compute the minimum response time of a transaction's task on a processor j .

- 1. Arrange the current tasks on processor j in ascending order of their local deadlines.*
- 2. Assign the highest priority to the new task. Assign priorities to other tasks based on this order, with the higher priority assigned to the task with the lower local deadline.*
- 3. Compute the new worst-case response times for the existing tasks based on this priority assignment.*
- 4. In the priority order, find the smallest position q such that for all tasks with position greater than q the worst-case response times are less than their corresponding local deadlines.*
- 5. Assign priority q to the new task and compute its worst-case response time. This worst-case response time is the minimum response time for the new task.*

This algorithm gives the minimum response time for a new task under the condition that for all existing tasks the local deadlines must be less than their corresponding periods. That is, the worst-case response time for each task does not exceed its inter-arrival time.

This method is proposed in [17] for establishing real-time channels. The only differences are as follows. (1) There is flexibility in selecting transactions for scheduling; in real-time communications, there is no such flexibility. The way the transactions are selected for scheduling affects the performance of the algorithm. Later, by simulation, the importance of ordering transactions for scheduling is shown. (2) In [17], the remaining slack time is divided equally among the tasks whereas here, the remaining slack time is divided among tasks in proportion to their minimum response times. A task with larger minimum response time receives more slack time. By giving more slack time to a task with larger minimum response time, the possibility that this task affects future unscheduled transactions is reduced.

transaction	deadlines on the first component	minimum response time (1)	deadlines on the second component	minimum response time (2)
1	1.33	1	2.66	2
2	6.12	5.5	8.88	8
3	0.38	0.25	4.62	3
4	7.70	7.5	12.3	12

Table 4.4 Deadline Assignment Using RTC

Table 4.4 shows how RTC works for the transactions defined in the Table 4.1. The algorithm selects transactions for scheduling based on the lowest slack time. In Table 4.4 transaction 1 has the lowest slack time and it is scheduled first. The minimum response time of this transaction on the first processor is 1 and on the second processor is 2. The remaining slack time for the transaction is 1 and this slack time is divided between the two tasks of

the transaction. The remaining slack time is divided in proportion to the minimum response time. Next, transaction 3 is selected for scheduling; this transaction has the lowest slack time after the transaction 1. The transaction is scheduled such that it does not affect the previous scheduled transactions; i.e., the previously scheduled transactions should not miss their assigned local deadlines. The minimum response times for this transaction are computed as 0.25 and 3 on processors 1 and 2, respectively. The remaining slack time of 1.75, is divided again in proportion to the minimum response time. This procedure is repeated for the two remaining transactions.

RTC might be used for handling dynamic transactions. Dynamic transactions are scheduled at run-time. The schedulability of a new arrived transaction can be tested using RTC and if the addition of the transaction does not affect deadlines of other transactions, the transaction will be accepted and scheduled based on the RTC algorithm. This capability of RTC has motivated us to consider it as one the heuristics for priority assignment problem.

(4) Worst Position (WP)

In this method, each processor is assumed to consist of a number of cells. Each cell represents a priority level and a task can be assigned to it. In each iteration the next cell with the highest response time (i.e., the *current worst-case position*) is determined. When tasks are independent the response time of a cell is the worst-case response time of any arbitrary task which is assigned to the cell. Note that response time of a cell is the same irrespective of which task is assigned to it. The response time of a cell, R , on an arbitrary processor j is computed using the iterative Equation 4.4. In this equation, $unschedule(j)$ is the set of all the unscheduled tasks on processor j and C and Q represent a task's execution time and period respectively. For finding the highest response time cell, the response times of all the lowest priority empty cells on all the processors are computed and the one with highest response time is selected. After determining the highest response time cell, an unscheduled

$$R^{n+1} = \sum_{\forall k \in \text{unschedule}(j)} \left\lceil \frac{R^n}{Q_k} \right\rceil C_k \quad (\text{EQ 4.4})$$

task is selected and is assigned to this cell. The selected task makes the flexibility of its transaction the highest. The algorithm terminates either after all cells are assigned or the next cell with highest response time cannot be assigned to any unscheduled task because the corresponding transaction misses its end-to-end deadline. Different steps of the algorithm are shown below.

1. Find the next cell with the highest response time. Assume this cell is located on processor j and the highest response time is equal to HRT .
2. Select a task T_k from the set of unscheduled tasks on processor j . Assign this task to the cell selected in (1) and find its transaction flexibility using the following equation. In this equation $DEADLINE$ shows the current deadline of the transaction corresponding to the selected task and $WORK$ shows the remaining execution time needed for the transaction. The initial value for $DEADLINE$ is the end-to-end deadline of the transaction and the initial value for $WORK$ is the summation of execution times of all the tasks belonging to the transaction.

$$FLEXIBILITY = \frac{DEADLINE - HRT - (WORK - PR(T_k))}{WORK - PR(T_k)} \quad (\text{EQ 4.5})$$

3. Do step (2) for all the unscheduled tasks on processor j . Select the task for which its corresponding transaction has the highest flexibility (task T_i) and assign it to the cell found in step (1). If flexibility is negative for all the tasks, there is no feasible solution and the algorithm terminates.
4. Current deadline of the transaction corresponding to the task selected in step (3) is changed to $DEADLINE - HRT$. The remaining work for the transaction is changed to $WORK - PR(T_i)$, where $PR(T_i)$ represents the execution time of task T_i .
5. If there are unscheduled tasks on any of the processors, go to step (1) else terminate.

Figure 4.1 shows the 8 steps of the algorithms for the workload defined in the Table 4.1. In the beginning all the cells are empty and the worst-case response time corresponds to the lowest priority cell in processor 2; this worst-case response time is equal to 12. The task of transaction 4 is assigned to this cell because transaction 4 has a higher flexibility than other transactions after assignment to this cell. In the next step, the worst-case response

time on processor 1 is 7.5 (same as the first step) and on processor 2 is 8. Because the worst-case response time is higher on processor 2, the next cell on processor 2 is selected to be assigned to an unscheduled task. The task belonging to transaction 2 is chosen because it has higher flexibility compared to the other transactions. The algorithm terminates after 8 steps when all tasks are assigned to cells.

WP introduces a different approach for finding a feasible solution. It starts from an empty schedule (i.e., no task is assigned to any priority level on any processor) and it tries to assign the tasks to priority levels one at a time, by considering timing constraints, until a feasible solution is found. In other words, the schedule is constructed step by step. In this approach, once a partial configuration violates any of the timing constraints, it cannot be rectified by assigning more tasks to the remaining priority levels. EQF and LM also follow the same concept except that the unit of construction in EQF is a processor and in WP is a task. In EQF, in each step, priorities are assigned to all the tasks running on a processor. In WP, by using a smaller unit of construction, more refinement is used in determining a schedule as compared to EQF and LM.

Figure 4.1 Priority Assignment Using WP

(5) HOPA Algorithm

The HOPA algorithm [12] is based on the distribution of the end-to-end deadline of each transaction among its tasks. Once each task is assigned an initial local deadline, deadline monotonic priorities are assigned in each processor and the worst-case response time is calculated for all the tasks and transactions. An analysis of the whole system is carried out and as a result of the analysis, new local deadlines are calculated. The iteration proceeds until either a feasible solution is found or the algorithm has gone through a specific number of iterations. The initial local deadlines are assigned in any way that preserves the end-to-end deadline. One possible assignment is to distribute the end-to-end deadline proportionally to the worst-case execution time of each transaction's task. A brief description of the HOPA algorithm is shown below. In [12], a heuristic method for calculating new local deadlines is proposed. In the heuristic the new local deadlines are calculated by redistributing the end-to-end deadlines among the transactions tasks. This redistribution takes into account the *excess of each task* (the difference between task's current local deadline and its worst-case response time), which intuitively, measures the distance that separates each task from schedulability. The new local deadlines are obtained as a function of two factors: on the one hand, the excess of each task relative to other tasks in the same transaction; on the other hand the amount of the excess on the resource to which the task belongs compared to excess on other resources. The excess on a resource is defined as the summation of task excesses for those tasks that belong to the resource.

- 1. Assign initial local deadlines.**
- 2. Assign priorities according to local deadlines. Priorities are assigned using deadline monotonic scheduling.**
- 3. Calculate worst-case response times for all tasks and transactions.**
- 4. If the worst-case response of all the transactions is less than their corresponding end-to-end deadline then the algorithm has found a feasible solution and it terminates.**

5. *Exit either after a given number of iterations or whenever two consecutive deadline assignments are identical. When the two consecutive deadline assignment are identical the algorithm would continue providing the same unfeasible solution.*
6. *Calculate new local deadlines based on a heuristic.*
7. *Go to (2)*

HOPA is considered in this research because it introduces a different approach for priority assignment problem. Rather than the construction approach used by WP, HOPA uses a transform method to find a feasible solution. It starts with an arbitrary priority assignment and transforms the initial assignment iteratively, going from one assignment to another until a feasible solution found. The sequence of transformations is based on a heuristic algorithm.

4.3 Simulation Model

In order to compare and study different priority assignment strategies, a simulation model is used and extensive experimentation is performed. For experiment reported in this chapter, a periodic flow shop model is assumed. Each transaction goes through a fixed set of processors and requires some amount of processing on each processor. Although this does not model any specific real-time application, it can provide useful information regarding strengths or weaknesses of different priority assignment strategies.

A set of preliminary experiments for getting insights into the behaviour the algorithms is done first. The results of these experiments are described in section 4.4. A more detailed investigation is presented in section 4.5. In all the experiments, a factor of interest is the number of transactions. Real-time systems often consist of numerous transactions and the heuristics should continue to show good performance with increasing number of transactions. Transactions require some processing time on each processor. In the preliminary experiments, a uniform distribution is used for modeling task's processing times (i.e., $pr(T_j) \sim U(0,1)$). On the other hand, each transaction t has a *flexibility*, $fl(t)$, that is defined

as follows. In this equation $dl(t)$ refers to the end-to-end deadline of the transaction and m is the number of processors in the flow shop.

$$fl(t) = \left(dl(t) - \sum_{j=1}^m pr(T_j) \right) / \left(\sum_{j=1}^m pr(T_j) \right) \quad (\text{EQ 4.6})$$

Intuitively the more flexible a transaction (higher $fl(t)$) the less stringent is its timing constraint. For the preliminary experiments, a normal distribution with a mean 5.0 and a standard deviation 1.0 is used for the flexibility of transactions (i.e., $fl(t) \sim N(\mu=5, \sigma=1)$). After generating the task execution times and flexibility for a transaction, the transaction deadline is computed as shown in Equation 4.16.

$$dl(t) = (1 + fl(t)) \times \sum_{j=1}^m pr(T_j) \quad (\text{EQ 4.7})$$

In this research, it is assumed that each transaction is periodic and the period of a transaction is the same as its deadline, but for initial experiments the periodicity is not taken into account. Periodicity can lead to several preemptions of a task by subsequent invocations of a higher priority task. In later experiments the effect of periodicity on the performance of the heuristics is explored.

4.4 Simulation Results

In this section the performance of the heuristics are compared. The analysis is based on generating a sufficient number of random samples (50000 problem instances are generated in each experiment) and determining which heuristic finds a larger number of feasible schedules. The algorithms are coded in the C language and they are tested by tracing them over a number of problem instances to ensure that they are behaving correctly. Applying any two different heuristics, say A and B, to random samples divides the samples into four disjoint subsets: (1) those samples that are solved by algorithm A but not B; (2) those samples that are solved by algorithm B but not A; (3) those samples that are solved by both algorithms; and (4) those samples not solved by either algorithm. The last subset can be

divided into another two disjoint subsets: (4a) those samples that do not have a feasible schedule; (4b) those that do have a feasible schedule.

The process of sampling from a given distribution for different system parameters and then discarding the unsolved instances may change the original distribution. For this reason, we avoid experiments that reject more than 10% of the generated problem instances. By this, we try to remain close to the original defined distribution for generating problem instances. The performance metric of interest is the *hit ratio* for the subsets 1, 2, and 3 which is defined as the number of samples in each subset with respect to the total number of samples for which schedules were found.

4.4.1 Simulation Results Without Considering Periodicity

Figure 4.2 shows the relative performance of algorithm EQF with respect to LM. In the figure, the X axis corresponds to the number of transactions and the Y axis shows the hit ratio: the portion of solution found by only EQF, only LM, and both algorithms (EQF & LM) relative to the number of solution found by all algorithms. The total number of solutions found would not be less than 90% of the total number of problem instances. In this figure as well as subsequent figures displaying the relative performance of the priority assignment algorithms the black bar is provided first, followed by a gray bar and then by a light gray bar. In the experiment captured by Figure 4.2, the number of processors in the flow-shop is two and the number of transactions is varied from 6 to 12. For any number of transactions, 50000 problem instances are generated and portion of solutions found by only EQF, only LM, and both algorithms (EQF & LM) are shown by three bars (from left to right, black bar for only EQF, gray bar for only LM, and light gray for EQF & LM). For example in Figure 4.2, when the number of transactions is 6 the portion of solutions found by only EQF and only LM is close to zero (hit ratio for only EQF and only LM is close to zero) and portion of solutions found by both algorithms is almost 1 (hit ratio for EQF & LM is close to 1). The results show that by increasing the number of transactions, the EQF

algorithm outperforms the LM algorithm; that is, by increasing the number of transactions, the portion of solutions found by only EQF is increased. This behavior is not surprising. The LM algorithm gives a larger proportion of the slack time to the tasks on the processor being scheduled and it does not consider the portion of the slack time that belongs to the future activities of transactions. This can result in poor decisions in early processors. For example, a transaction with considerable work in the future but much less work on the processor being scheduled may get the lowest priority because it is assigned all the slack. Therefore, the transaction consumes all of its slack time and leaves little slack time for future activities.

Figure 4.2 The Performance of EQF Versus LM for a 2 Processor Flow Shop

By increasing the number of processors, the performance gap between the algorithms becomes wider. Figure 4.3 shows that by increasing the number of processors in the flow-shop the EQF performs much better than LM. This confirms that poor decisions at early

stages do not leave enough slack for the final stage tasks. As a result, it is more likely that transactions miss their deadlines. In Figure 4.3, when the number of transactions is equal to fourteen, almost all the feasible schedules are found by the EQF algorithm; 50000 samples are generated and feasible schedules for close to 90% of them are computed by the EQF algorithm. For the remaining 10% nothing can be said. Thus although a substantial portion of samples have feasible schedules, the LM algorithm is not successful in finding even one of them.

Figure 4.3 The Performance of the EQF Versus LM for a 10 Processor Flow Shop

It is expected that for those samples for which feasible schedules are found by both LM and EQF, EQF gives a higher remaining slack time when compared to LM. To check this, two random variables are defined: S_{lm} corresponds to the remaining slack time in the LM algorithm and S_{eqf} corresponds to the remaining slack time in the EQF algorithm. For the case of 10 transactions and 15 processors, samples are taken for the random variable $S_{eqf} - S_{lm}$ and its PDF is shown in Figure 4.4. It can be seen that EQF algorithm has a higher remaining slack time when compared to LM.

Figure 4.5 shows the result for the same experiment when the number of transaction is reduced to 5. In this experiment the difference between two algorithms is less. By reducing number of transactions, the poor priority assignment does not show itself as strongly.

A problem in the EQF algorithm arises when both long and short transactions exist together. A short transaction might be as short as a transaction with only one task. In this situation, using EQF, a short transaction's task receives the major portion of the transaction's slack time. On the other hand, a long transaction's task, competing with the short transaction's task, receives a small portion of slack time and consequently it has a higher priority. This happens in the situation where if the short transaction's task misses its assigned local deadline, there is a good chance that the transaction misses its end-to-end deadline. In other words, short and long transactions do not compete fairly and the algorithm is biased against short transactions. To examine this phenomena, an experiment was run and the ratio of short transactions to long transactions was changed. The experiment was done for a 10 processor flow-shop with 10 transactions. The probability of existence of short transactions with a single task is changed. To make this phenomena more observable it is assumed that the tasks of the short transactions exist only on the first processor in the flow-shop. With this assumption it is more likely that short transactions get lower priority compared to long transactions. The probability of the existence of short transactions is changed from 0 to 1 and for each case, 50000 samples are generated. Table 4.5 shows the results. Table 4.5(a) shows the result when the EQF algorithm is applied. Table 4.5(b) corresponds to the case that transactions are divided into two classes; i.e., short and long transactions, and higher priority is assigned to the short transaction class. It can be seen that the number of schedules found in Table 4.5(b) is more than the Table 4.5(a). The results of this experiment show that because of the co-existence of short and long transactions, the EQF algorithm may not be able to find a feasible priority assignment when feasible schedules exist.

Figure 4.4 PDF of the $(S_{eqf} - S_{lm})$ for 10 Transactions and a 15 Processor Flow Shop

Figure 4.5 PDF of the $(S_{eqf} - S_{lm})$ for 5 Transactions and a 15 Processor Flow Shop

Portion of short transactions	Number of feasible schedules
0	50000
0.1	48004
0.2	46268
0.3	44725
0.4	43406
0.5	41957
0.6	40653
0.7	39852
0.8	38753
0.9	38043
1.0	37151

EQF

(a)

Portion of short transactions	Number of feasible schedules
0	50000
0.1	49999
0.2	49999
0.3	49994
0.4	49961
0.5	49836
0.6	49456
0.7	48644
0.8	46739
0.9	43245
1.0	37151

Higher priority for local task

(b)

Table 4.5 The Effect of Local Tasks in the EQF Algorithm

Next, the performance of the EQF is compared to RTC; the results are shown in Figure 4.6 and 4.7 for two and ten processor flow-shops, respectively. It can be concluded that for a large number of processors, EQF outperforms RTC. For a two processor flow-shop, as it can be seen from Figure 4.6, there are a significant number of cases in which EQF is not successful but RTC succeeds in generating a feasible schedule.

One reason for the superiority of EQF over RTC is how priority is assigned to the tasks with short execution times. In EQF, a small slack time is assigned to these tasks and consequently they receive high priority; these tasks, because of their short execution times do not significantly affect lower priority tasks. In the RTC method, there is no guarantee that these tasks get lower priorities; in this method, the worst-case response time for these tasks can be much more than their required execution times. Moreover, in RTC, it is impossible to compensate for poor priority assignment on succeeding processors, thus, in

RTC, by increasing the number of processors, the problem of splitting the remaining slack time among different tasks of a transaction becomes more significant. A drawback of RTC is that deadlines assigned to different tasks of a transaction remain fixed. In RTC for satisfying the end-to-end deadlines of transactions, the fixed local deadline assigned to the tasks must be met. This is not a necessary requirement and can degrade the performance of the algorithm.

Figure 4.6 The Performance of the EQF Versus RTC for a 2 Processor Flow Shop

Figure 4.7 The Performance of the EQF Versus RTC for a 10 Processor Flow Shop

The performance of the RTC algorithm is better if transactions are scheduled based on the lowest slack time. If a transaction with a small slack time is scheduled last, it is very likely that it would be affected by previously scheduled transactions and consequently, because of the small slack time miss its deadline. Figure 4.8 shows the effect of lowest slack time policy on the performance of RTC. RTC1 selects transactions randomly for scheduling. Figure 4.8 shows that as the number of transactions increases RTC1 is not able to solve a substantial number of problem instances.

RTC divides the remaining slack time of a transaction among its tasks in proportion to the tasks minimum response times. In [12], the remaining slack time is divided equally among tasks. Figure 4.9 shows the performance difference between these two policies for a ten

processor flow shop. In this figure, RTC2 represents the algorithm that divides the remaining slack time equally among a transaction's tasks. By splitting the remaining slack time among tasks in proportion to their minimum response time, more slack time is given to a task with a large minimum response time and less slack time is given to a task with a small minimum response time; consequently the probability that a task with a large minimum response time affects subsequent transactions becomes smaller. Also giving less slack time to a task with a small minimum response time, saves slack time for the other tasks of transaction; a task with a small minimum response time does not contribute much to the response time of other tasks running on the same processor.

Figure 4.8 The Performance of RTC for a 10 Processor Flow Shop when Transactions are Selected for Scheduling Randomly

Figure 4.9 The Performance of RTC for a 10 Processor Flow Shop when Remaining Slack Time is Divided in Proportion to Tasks Minimum Response Time

Figures 4.10 and 4.11 shows the performance of EQF versus WP for 2 and 10 processor flow shops. In Figure 4.10, a substantial proportion of feasible solutions is found by WP. For example, when number of transactions is 12 close to 30% of the feasible solutions are found by WP only. Figure 4.11 shows that by increasing the number of processor to 10, both algorithms are successful in finding feasible solutions. This phenomena can be explained as follows.

By increasing the number of processors, the maximum achievable utilization on each processor will drop. The drop is necessary to satisfy the end-to-end deadline of the transactions. When utilization on processors is low, WP does not properly differentiate among candidate tasks for the next worst-case position. It can also happen that a short task (a task

with a low execution time) becomes assigned to the worst-case position. Assigning a short task to a low priority position is not a good decision. Assigning a high priority to a short task does not substantially affect the response times of other tasks and also saves slack time for the transaction corresponding to the short task.

The performance of HOPA is also compared to WP. Figure 4.12 and 4.13 show the results for two and ten processor flow shops. In both cases, there are lot of cases in which WP can find feasible solutions but HOPA does not succeed.

Comparing Figures 4.10 and 4.12 (i.e when the number of processors is 2), it is observed that HOPA performs better than EQF; when the number of transactions is 12 the number of solutions found by only WP when compared to EQF is close to 40% and when compared to HOPA is less than 20%. However, when the number of processors is increased to 10 (Figures 4.11 and 4.13), it is observed that for many problem instances, both EQF and WP can find feasible solutions that HOPA cannot. For example, in Figure 4.11, when the number of transactions is 14, both EQF and WP are able to find feasible solutions for generated problem instances but HOPA is not able to find solutions for 20% of problems (Figure 4.13). This shows that HOPA does not perform well when the number of processors is increased.

Figure 4.10 Performance of EQF Versus WP for 2 Processor Flow Shop

Figure 4.11 Performance of EQF Versus WP for 10 Processor Flow Shop

2

PM-1 3 1/2"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1"10a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

Figure 4.12 Performance of HOPA Versus WP for 2 Processor Flow Shop

Figure 4.13 Performance of HOPA Versus WP for 10 Processor Flow Shop

In summary, in this section it is shown that: (1) the EQF algorithm outperforms the LM algorithm especially for large numbers of transactions and processors; (2) EQF has a better performance than RTC for a large number of processors; (3) in RTC by dividing the remaining slack time of a transaction in proportion to minimum response time of its tasks gives a better performance compared to dividing the slack time equally among tasks; (4) in RTC selecting transactions for scheduling based on lowest slack time gives a better performance compared to selecting them arbitrarily; (5) WP is able to find feasible priority assignments for many problem instances where EQF is not successful; and (6) WP can find feasible priority assignments for many problem instances where HOPA is not successful.

4.4.2 Simulation Results When Considering Periodicity

In this section, the effect of periodicity on the performance of the heuristics is taken into account. The major reason for the inadequacy of the LM algorithm is its poor decisions at early stages. This algorithm, by giving all the remaining slack to the tasks under scheduling, does not assign enough slack time for the tasks yet to be scheduled. If a decision at an early stage is poor, periodicity will make it worse and amplify the undesirable effect of the phenomena. So, it is expected that in the face of periodicity, the EQF algorithm shows a better performance compared to the LM algorithm.

The effect of periodicity can be seen by comparing Figures 4.14 and 4.15. In Figure 4.15 when the number of transactions is equal to 10, but periodicity is not in effect, close to 80% of the feasible schedules are found by both algorithms. On the other hand in Figure 4.14, which corresponds to periodic transactions, close to 60% of the schedules are found by both algorithms and almost all the remaining ones found by EQF. This confirms the hypothesis that under periodicity, the performance gap between LM and EQF gets larger.

The performance of EQF is also compared to that of RTC for the three processor flow-shop. The results are shown in Figure 4.16 and 4.17. In the case that the number of transactions is 10 and the periodicity is not in effect, RTC is able to find approximately 90% of feasible schedules but by introducing periodicity the hit ratio for RTC drops to approximately 80%. The results show that in RTC, the problem of splitting remaining slack time among the tasks of a transaction becomes more important when transactions are periodic.

In Figures 4.18 and 4.19 the effect of periodicity on the relative performance of EQF and WP are captured. When the number of transactions is 11 and periodicity is not in effect, EQF is able to find close to 100% of the feasible solutions but by introducing periodicity the hit ratio of EQF drops approximately to 80%. The results show that WP performs better than EQF when periodicity is in effect.

In Figures 4.20 and 4.21 the effect of periodicity on the relative performance of WP and HOPA is studied. When the number of transactions is 11, by introducing periodicity (Figure 4.21) HOPA is not able to find close to 8% of the feasible solutions. This shows WP performs better than HOPA when periodicity is in effect.

Figure 4.14 Performance of EQF Versus LM for 3 Processor Periodic flow Shop

Figure 4.15 Performance of EQF Versus LM for 3 Processor Non-Periodic Flow Shop

Figure 4.16 Performance of EQF Versus RTC for 3 Processor Periodic Flow Shop

Figure 4.17 Performance of EQF Versus RTC for 3 Processor Non-Periodic Flow Shop

Figure 4.18 Performance of EQF Versus WP for a 3 Processor Periodic Flow Shop

Figure 4.19 Performance of EQF Versus WP for a 3 Processor Non-Periodic Flow Shop

Figure 4.20 Performance of WP Versus HOPA for a 3 Processor Periodic Flow Shop

Figure 4.21 Performance of WP Versus HOPA for a 3 Processor Non-Periodic Flow Shop

4.4.3 Summary of Basic Results

The basic simulation results have led to the following conclusions.

- EQF performs better than LM especially for a large number of processors.
- EQF performs better than RTC when increasing the number of processors.
- In RTC, a better performance is achieved by selecting a transaction based on lowest slack time first policy.
- A better performance can be achieved in RTC by splitting the remaining slack time in proportion to minimum response times of tasks, rather than splitting the remaining slack time in equal proportion.
- The WP algorithm is able to find feasible solutions for many cases in which EQF is not successful.
- The WP algorithm is able to find feasible solutions for many cases in which HOPA is not successful.

4.5 The Effect of Workload Parameters

In section 4.4, some general conclusions on the behavior of the algorithms are made and it is shown through simulation that the algorithms introduced in this research can improve the coverage of existing algorithms. In this section, the effect of different workload parameters on the relative performance of algorithms is studied. The parameters that are investigated are listed below.

- Mean of task execution time (ME)
- Coefficient of variation of task execution time (CVE)

- Mean of transaction flexibility (MF)
- Coefficient of variation of transaction flexibility (CVF)
- Number of transactions (NT)

Gamma distribution is used for generating task execution times and flexibility for transactions. By using a Gamma distribution, a high variation in both task execution time and transaction flexibility can be achieved. To study the effect of each parameter on the relative performance of any two algorithms different experiments are designed; the experiments are defined in Table 4.6. To limit the result space, we have decided to keep the number of processors fixed for all the experiments. Each row in Table 4.6 represents one experiment; in this table, 24 different experiments are specified. Experiments are divided into 4 groups; each group (shown as a separate table) studies the effect of one of the parameters indicated on the left hand side of each table which is varied over a reasonable range. Each of these experiments is run four times to compare the relative performance of four different pairs of algorithms. The algorithms pairs are [WP, EQF], [WP, HOPA], [EQF, RTC], and [EQF, LM]. In the following sections bar graphs are used to report experiments illustrating major differences in the relative performances; however all the results are discussed and the complete set of numeric results is given in Appendix A.

4.5.1 The Effect of the Mean of Task Execution Time

By changing the ME, it is observed that the number of feasible solutions found by algorithms does not change and the relative performance of algorithms also remained the same. Changing the ME does not change the structure of the problem instances. Priorities assigned to tasks by any of the algorithms will remain the same. Because ME has only a scaling effect, it is not considered as a factor in later discussions.

CVF IS CHANGED FROM 1.0 TO 2.0		0.0	2.5	1.0	5	3
		0.25	2.5	1.0	5	3
		0.5	2.5	1.0	5	3
		0.0	5.0	1.0	10	3
		0.25	5.0	1.0	10	3
		0.5	5.0	1.0	10	3
CVF IS CHANGED FROM 0.0 TO 0.5	0.5		2.5	1.0	5	3
	1.0		2.5	1.0	5	3
	2.0		2.5	1.0	5	3
	0.5		5.0	1.0	10	3
	1.0		5.0	1.0	10	3
	2.0		5.0	1.0	10	3
MF IS CHANGED FROM 1.5 TO 3.5	0.5	0.0		1.0	5	3
	2.0	0.0		1.0	5	3
	0.0	0.5		1.0	5	3
MF IS CHANGED FROM 4.0 TO 6.0	0.5	0.0		1.0	10	3
	2.0	0.0		1.0	10	3
	0.0	0.5		1.0	10	3
NT IS CHANGED FROM 1 TO 6	0.5	0.0	2.5	1.0		3
	2.0	0.0	2.5	1.0		3
	0.0	0.5	2.5	1.0		3
NT IS CHANGED FROM 7 TO 12	0.5	0.0	5.0	1.0		3
	2.0	0.0	5.0	1.0		3
	0.0	0.5	5.0	1.0		3

Table 4.6 Experiments for Flow Shop

4.5.2 The Effect of the Coefficient of Variation of Task Execution Time

Figures 4.22, 4.23, 4.24 and 4.25 show the relative performance of algorithms when NT is 10, CVF is 0.5 and MF is 5.0. The CVE is varied from 0 to 2.0. Figure 4.22 and 4.23 show that at low levels of CVE, algorithm WP can find solutions in many cases in which EQF and HOPA are not successful. Results also show that HOPA is able to find more solutions than EQF. Comparing the performance of EQF and RTC, many solutions are found by only EQF at low values of CVE (Figure 4.24). Figure 4.25 shows that at low levels of CVE, EQF and LM behave the same and most solutions are found by both algorithms. By increasing CVE, EQF outperforms LM.

Increasing the CVE, the relative performance of any pair of algorithms tends to get closer. This shows that by increasing the CVE, the problem instances become less challenging and easier to solve. Also increasing CVE, increases the number of problem instances for which the utilization on one of the processors exceeds 1.0.

4.5.3 The Effect of the Mean of Transaction Flexibility

Figure 4.26, 4.27, 4.28 and 4.29 shows the effect of MF when NT is 10, NP is 3, CVE is 0 and CVF is 0.5. MF is changed from 5.0 to 6.0. Figure 4.26 and 4.27 shows that at low levels of the MF, when problem instances are more difficult, WP can find solutions for many cases in which HOPA and EQF are not successful. The figures also show that HOPA is more successful in finding solutions when compared to EQF. Figure 4.28 compares the performance of EQF and RTC. At low values of MF, RTC shows a very poor performance. By increasing the MF and consequently making problem instances easier, RTC shows a better performance. Figure 4.29 compares the relative performance of EQF and LM; the performance of the two algorithms is the same over the range of variation of MF.

Figure 4.22 The Effect of CVE on Relative Performance of EQF and WP

Figure 4.23 The Effect of CVE on Relative Performance of HOPA and WP

Figure 4.24 The Effect of CVE on Relative Performance of EQF and RTC

Figure 4.25 The Effect of CVE on Relative Performance of EQF and LM

Figure 4.26 The Effect of MF on Relative Performance of EQF and WP

Figure 4.27 The Effect of MF on Relative Performance of HOPA and WP

Figure 4.28 The Effect of MF on Relative Performance of EQF and RTC

Figure 4.29 The Effect of MF on Relative Performance of EQF and LM

4.5.4 The Effect of Coefficient of Variation of Transaction Flexibility

Figures 4.30, 4.31, 4.32 and 4.33 show the effect of CVF on the relative performance of algorithms when NT is 10, NP is 3, CVE is 0.5 and MF is 5.0. Figures 4.30 and 4.31 show that as CVF increases, WP finds more solutions than EQF and HOPA. In Figures 4.32 and 4.33, by increasing CVF from 0 to 0.3, the number of solutions found by only EQF tends to increase. In the same figures, for CVF equal to 0.4 and 0.5, the number of solutions found by only EQF tends to remain constant or even drop. The reason for this is a drop in the number of solutions found by the algorithms which affects the original distribution of the flexibility factor. In other words, neither of the algorithms are able to solve difficult problem instances and by discarding these problems the performance of algorithms are compared over a simpler problem set. Comparing over a simpler problem set will cause the relative performance of algorithms to get closer.

4.5.5 The Effect of Number of Transactions

Figures 4.34, 4.35, 4.36 and 4.37 show the effect of NT when NP is 3, CVE is 0 and CVF is 0.5. In all the experiments, by increasing the NT, problem instances become more difficult. Figures 4.34 and 4.35 show that by increasing NT, the number of solutions found by only WP tends to increase. Figures 4.34 and 4.35 also show that HOPA is more successful in finding feasible solutions compared to EQF. Figure 4.36 compares the performance of EQF and RTC. By increasing NT most of the solutions are found only by EQF. Figure 4.37 does not show difference between EQF and LM as the number of transaction has increased. When the variation in task execution time is 0, EQF and LM behave the same and the same priorities are assigned to tasks using either of the algorithms.

Figure 4.30 The Effect of CVF on Relative Performance of EQF and WP

Figure 4.31 The Effect of CVF on Relative Performance of HOPA and WP

Figure 4.32 The Effect of CVF on Relative Performance of EQF and RTC

Figure 4.33 The Effect of CVF on Relative Performance of EQF and LM

Figure 4.34 The Effect of NT on Relative Performance of EQF and WP

Figure 4.35 The Effect of NT on Relative Performance of HOPA and WP

Figure 4.36 The Effect of NT on Relative Performance of EQF and RTC

Figure 4.37 The Effect of NT on Relative Performance of EQF and LM

4.5.6 Summary of Simulation Results

In this section the relative performance of algorithms is discussed based on the experiments defined in Table 4.6.

With high values of CVE, the experiments do not reveal any differences between algorithms. The reason for this is that the higher variation of task execution time either produced problems that are unsolvable (because of utilization greater than 1 on one or more processors), or problems that are easy to solve. The results reported below demonstrate the difference in performance of the policies when CVE was low (almost 0) to moderate (0.5).

In the notes below, "algorithm A outperformed algorithm B" means that A is able to find a substantial number of solutions that B is not, and the opposite is not true.

1) WP Versus EQF

When there is a noticeable performance difference, WP outperforms EQF. This occurs for low values of CVE, under higher loads (greater number of transactions or smaller values for the mean of flexibility, or larger values for the coefficient of variation of flexibility). Tables A.7, A.9, A.10, and A.12 (Appendix A) show this phenomena when load is increased by increasing number of transactions, Tables A.13, A.15, A.16 and A.18 when load is increased by decreasing the mean of flexibility, and Tables A.19 and A.22 when load is increased by increasing the coefficient of variation of flexibility.

2) WP Versus HOPA

When there is a noticeable performance difference, WP outperforms HOPA. This occurs particularly under heavy load when the number of transactions is high or the mean of flexibility is low, or the coefficient of variation of flexibility is high. This can be seen from Tables A.31, A.33, A.34 and A.36 when the number of transactions is high, from Tables A.37, A.39, A.40 and A.42 when the mean of flexibility is low, and from Tables A.43 and

A.44 when the coefficient of variation of flexibility is high. From the experiments, it can be concluded that although HOPA is outperformed by WP under heavy load conditions, it is more successful than EQF in finding feasible solutions for the generated problem instances.

3) EQF Versus RTC

When there is a noticeable performance difference, EQF outperformed RTC. The degree to which it is outperformed (some, versus overwhelming), increased with higher loads. Tables A.55, A.57, A.58, and A.60 shows the effect of increasing load when the number of transactions is increased and Tables A.61, A.63, A.64, and A.66 shows the effect of increasing load when the mean of the flexibility is decreased. The effect of increasing load by increasing the coefficient of variation of flexibility is shown in Tables A.67 and A.70.

4) EQF Versus LM

When there is a noticeable performance difference, EQF outperforms LM. LM has the characteristic of doing as well as EQF at very low values of CVE. However, otherwise, it performs quite poorly, even at high CVE. In Tables A.81, A.84, A.87 and A.90 load is increased either by increasing the number of transactions or decreasing the mean of flexibility. In all these tables the performance difference between EQF and LM is unnoticeable because CVE is 0.

None of the algorithms performs the best for all problem instances. However, In general, WP covered the greatest number of solutions, since it largely subsumed EQF which in turn largely subsumed RTC and LM. The greatest coverage might be achieved by WP with HOPA and EQF.

Chapter 5

Extending Priority Assignment Algorithms to More Complex Problem Domains

5.1 Introduction

In this chapter the EQF, RTC and WP algorithms, introduced in Chapter 4, are extended to more complex problem domains that include revisiting transactions or transactions with subtransactions. Extensions to the algorithms are described and illustrative examples are given by applying the extended algorithms to problem instances shown in Figures 5.1 and 5.2. The effect of revisiting and subtransactions on the relative performance of algorithms are studied. Also, algorithms are used to assign priorities to different tasks in the signal processor module of a towed array sonar system.

In Figure 5.1, transaction t_2 is a complex transaction (a transaction with subtransactions). The period and end-to-end deadline of the transaction are equal to 8. The complex transaction consists of two subtransactions. The first subtransaction goes through the tasks $T1$, $T3$, $T5$, and $T10$, and the second subtransaction goes through the tasks $T1$, $T6$, $T7$, and $T10$. The worst-case execution times of all the tasks are equal to 1. In this figure transactions t_1 and t_3 are simple transactions. A complex transaction is described by a set. Each element in the set corresponds to one of the subtransactions and is the sequence of tasks traversed by the subtransaction. For example, transaction t_2 is described by $\{[T1, T3, T5, T10], [T1, T6, T7, T10]\}$

Figure 5.2 shows an example of a revisiting transaction. The revisiting transaction goes through the tasks $T1$, $T4$, $T2$, and $T5$ and has visited processor $P2$ twice. Each of the tasks

$T1$ and $T2$ corresponds to one of the visits. The worst-case execution time for the tasks $T1$, $T4$, $T2$, and $T5$ are 3, 4, 1, and 2 respectively. The period and deadline of this transaction are equal to 20. In this figure transaction $t2$ is a simple transaction consisting of one task ($T3$) running on processor $P2$ and with period and deadline equal to 8. The worst-case execution time of task $T3$ is equal to 5. Using the same notation as transaction with subtransactions, the revisiting transaction $t1$ can be shown as the set $\{[T1.T4.T2.T5]\}$.

Figure 5.1 Transaction with Subtransactions

Figure 5.2 Revisiting Transaction

5.2 Extending RTC to More Complex Problem Domains

The extension of RTC (described in Chapter 4) to handle revisiting transactions and transactions with subtransactions is described. In the case of revisiting transactions the issue of computing the minimum response time of the revisiting transaction on the revisited processor is discussed and for transactions with subtransactions an algorithm for finding the local deadline of a task is introduced. To handle transactions with subtransactions, RTC needs to be extended in order to (1) map complex transaction end-to-end deadlines to subtransaction end-to-end deadlines and, (2) assign local deadlines to the tasks that are shared by several subtransactions.

5.2.1 Revisiting Transactions

In order to extend RTC to support revisiting transactions, the problem of finding the minimum response-time of a revisiting transaction on a processor must be solved. The problem can be stated as follows.

Assume: (1) the set $M = \{T_i = (C_i, Q_i, D_i) \mid i = 0, \dots, k-1\}$ shows a set of tasks running on processor j under deadline monotonic scheduling (C_i , Q_i , and D_i are the task's execution time, period, and deadline respectively) and (2) each task of the set $N = \{T_k, T_{(k+1)}, \dots, T_{(k+r-1)}\}$ corresponds to a visit of a revisiting transaction on processor j . The problem is how to schedule the tasks in set N such that the tasks in set M do not miss their deadlines and also $\sum_{i=k}^{k+r-1} R_i$ is minimized. R_i is the response time of task i .

Before studying the problem of minimizing the response time of revisiting transactions, an example is presented (see Figure 5.2). Assuming $T3$ on $P2$ has already been scheduled, the minimum response time of transaction $t1$ on $P2$ is required. Compared to task $T3$, tasks $T1$ and $T2$ can have different priorities; i.e., both tasks $T1$ and $T2$ can have a higher priority than $T3$, one of them can have a higher priority than $T3$ or both of them can have priorities

less than $T3$. The question is: which combination minimizes the response time of transaction $t1$ without causing transaction $t2$ to miss its deadline? If both $T1$ and $T2$ have a higher priority than $T3$, the worst-case response time of $T3$ will be equal to 9 and transaction $t2$ will miss its deadline. If either of tasks $T1$ or $T2$ is assigned a priority lower than $T3$, then the deadline of task $T3$ is satisfied. The response time of transaction $t1$ on $P2$ in either of these cases is equal to 9 which is the minimum response time of $t1$ on $P2$.

To the best of our knowledge, finding the minimum response time of a revisiting transaction is not discussed in the literature. Although, it is possible to introduce algorithms to approximate the minimum response time, they are not expected to yield good results, therefore extension of RTC for revisiting is not explored further.

5.2.2 Transactions with Subtransactions

In this section an algorithm to extend RTC is introduced. The algorithm assigns local deadlines to the tasks of complex transactions. Each subtransaction of the complex transaction is treated as a simple transaction with the end-to-end deadline equal to its corresponding complex transaction. Different subtransactions of a complex transactions can share the same tasks (i.e., forking and joining tasks). The algorithm RTC is extended to handle the shared tasks. The different steps of the extended algorithm are described below. The algorithm describes the scheduling of a complex transaction.

- 1. Select the next subtransaction from the set of subtransactions representing the complex transaction.***
- 2. For each of the tasks belonging to the subtransaction selected in (1) for which the local deadline is not assigned, find the task minimum response time on the processor on which the task is running.***
- 3. Find the minimum response time for the subtransaction selected in (1) by summing up the task minimum response times found in step (2) with local deadlines that have already been assigned to the subtransaction tasks.***

4. Find the remaining slack time by subtracting the transaction's end-to-end deadline from the minimum response time found in step (3). If the remaining slack time is negative the algorithm cannot find a feasible solution.
5. Divide the remaining slack time among the subtransaction's tasks for which a local deadline is not assigned. The remaining slack time is divided in proportion to a task's minimum response time.
6. Go to (1) until all the subtransactions are scheduled.

In the following example the above algorithm is used to assign priorities to the transactions tasks shown in Figure 5.1. Assume transaction $t1$ is scheduled using RTC. Because transaction $t1$ is the first transaction selected for scheduling, its minimum response time on processor $P2$ and $P3$ is the same as the *WCET* (Worst-Case Execution Time) of tasks $T2$ and $T4$. The remaining slack time for the transactions is 2 which is divided between the tasks in proportion to their minimum response times. Each task is assigned half of the remaining slack time and consequently the local deadlines for both tasks is the same and equal to 3. The next transaction selected for scheduling is complex transaction $t2$. The different steps of the algorithm when applied to this problem instance are described below. The numbers show the different steps of the algorithm.

1. The subtransaction $\{T1, T3, T5, T10\}$ is selected.
2. Local deadlines of all the tasks $T1, T3, T5, T10$ are not determined yet. The minimum response time for task $T1$ is equal to its *WCET* which is 1. The minimum response time for task $T3$ is 1. Note that task $T3$ can get the highest priority without causing task $T2$ to miss its deadline. With the same reasoning the minimum response time for task $T5$ is also 1. The minimum response time for task $T10$ is also 1.
3. By adding the calculated minimum response times for the tasks, the minimum response time for the subtransaction $\{T1, T3, T5, T10\}$ is 4.
4. The remaining slack-time for the subtransaction is end-to-end deadline minus the subtransaction's minimum response time which is 4.
5. Because the minimum response times for all the tasks are the same, the remaining slack-time is divided equally among tasks and the local deadlines for all the tasks are the same and equal to 2.
1. The subtransaction $\{T1, T6, T7, T10\}$ is selected.
2. For tasks $T1$ and $T10$ local deadlines have already been assigned. The minimum response times for the tasks $T6$ and $T7$ are equal to 1.

3. *The minimum response time for the subtransaction is the summation of minimum response times for tasks T6 and T7 and local deadlines for the tasks T1 and T10. This minimum response time is equal to $(2+1+1+2)=6$.*
4. *The remaining slack-time is the end-to-end deadline minus the subtransaction's minimum response time which is 2.*
5. *The remaining slack-time is divided between tasks T6 and T7. Because the minimum response time for both tasks is the same, the remaining slack-time is divided equally and each task receives a slack-time equal to 1. The local deadlines for these two tasks are the same and equal to 2.*

The last transaction to be scheduled is the simple transaction t_3 . The minimum response time of task T8 is 4. Note that this task cannot get a higher priority than T6 because task T6 would miss its deadline. The minimum response time for task T9 is 3. Task T9 also cannot get a higher priority than T7. If a higher priority is assigned to T9, task T7 would miss its local deadline. Consequently, the minimum response time for transaction t_3 is 7 which is beyond the end-to-end deadline of the transaction. So, no feasible priority assignment is found for the given problem instance.

The following scenario illustrates a potential problem with the extended RTC algorithm. If the algorithm starts with subtransaction $[T1, T3, T5, T10]$ and processors P1, P2, P3 and P4 are lightly loaded then the remaining slack time for this subtransaction can be high and substantial slack is given to tasks T1 and T10. If subtransaction $[T1, T6, T7, T10]$ is selected next for scheduling and processors P5 and P6 are highly loaded then it is very likely that this subtransaction would not meet the transaction end-to-end deadline, thus resulting in a failure to schedule the complex transaction. In other words, the summation of local deadlines for tasks T1 and T10 with minimum response times for tasks T6 and T7 is greater than the complex transaction's end-to-end deadline. Note that local deadlines for T1 and T10 are determined during the scheduling of transaction $[T1, T3, T5, T10]$.

5.3 Extending EQF to more Complex Problem Domains

The extensions of EQF (described in Chapter 4) to handle revisiting transactions and transactions with subtransactions are described. In the case of revisiting transactions, EQF policy is used for assigning local deadlines to tasks and the worst-case response time of tasks are computed using Equation 3.10 (described in Chapter 3) To solve transactions with subtransactions, EQF is extended in order to (1) map complex transaction end-to-end deadlines to subtransactions end-to-end deadlines and, (2) handle tasks that are shared by several subtransactions.

5.3.1 Revisiting Transactions

In the case of revisiting transactions, the only change made in EQF is using a different algorithm for computing worst-case response times for tasks. Equation 3.10 is used instead of Equation 3.2 to compute tasks response times. The extended EQF algorithm is shown below.

- 1. Set current deadline of each transaction to its end-to-end deadline.*
- 2. Determine processors that have unscheduled tasks.*
- 3. Among the processors determined in step (2) select the one with the highest utilization. Name this processor j .*
- 4. Use EQF policy to assign local deadlines to the tasks allocated to processor j (EQF policy is described in Chapter 4).*
- 5. Use local deadlines to assign priorities to the tasks on processor j . The task with lower deadline has higher priority.*
- 6. Use Equation 3.2, introduced in Chapter 3, to compute the worst-case response time of tasks on processor j .*
- 7. For each task on processor j , subtract the task's worst-case response time from the corresponding transaction's current deadline.*
- 8. If the current deadline of any transaction is negative, the algorithm is not able to find a feasible solution.*
- 9. Go to (2) until all the tasks are scheduled.*

The following example shows how EQF can be used to assign priorities to the tasks when revisiting transactions exist in the system. In this example EQF is applied to the problem instance shown in Figure 5.2.

- 1. The current deadlines of transactions $t1$ and $t2$ are 20 and 8 respectively.*
- 2. Processors $P1$, $P2$, and $P3$ need to be scheduled.*
- 3. Processor $P2$ is selected. It has highest utilization among $P1$, $P2$, and $P3$.*
- 4. Using EQF policy the local deadlines for tasks $T1$, $T2$, and $T3$ are 6, 2 and 8 respectively.*
- 5. Task $T2$ has the highest priority and task $T3$ has the lowest priority.*
- 6. The worst-case response time for task $T1$ and $T2$ is the same as their worst-case execution times and is equal to 3 and 1 respectively. The worst-case response time for $T3$ is 9.*
- 7. The current deadline for transactions $t1$ and $t2$ are 16 and -1 respectively.*
- 8. The current deadline for transaction $t2$ is negative, therefore the algorithm is not able to find a feasible solution.*

5.3.2 Transactions with Subtransactions

In this section an algorithm to extend EQF to solve transactions with subtransactions is introduced. The algorithm considers all the subtransactions of a complex transaction as simple transactions with the same end-to-end deadline as the complex transaction. The local deadlines for tasks are computed based on the EQF algorithm as explained in Chapter 4. If a task is shared by several simple transactions, the local deadline of the task is determined by that simple transaction which requires the lowest local deadline for the task. The algorithm is explained below.

- 1. Determine all the subtransactions belonging to each complex transactions.*
- 2. Assign the end-to-end deadline of each complex transaction to its subtransactions.*
- 3. Select an unscheduled processor (an unscheduled processor is the one in which priorities are not assigned to the tasks running on it). The selected processor has the highest utilization among all the unscheduled processors.*

4. *Assign local deadlines to the tasks allocated to the processor using EQF algorithm. If a task is used by several subtransactions the subtransaction that produces the lowest local deadline is used by the EQF algorithm to assign the local deadline to the task.*
5. *Find the worst-case response times of the tasks on the selected processor using deadline monotonic priority assignment.*
6. *Subtract the worst-case response time of the tasks from their corresponding transaction's or subtransaction's current deadline. Note that if a task is used by different subtransactions of a complex transaction then the worst-case response time of the task is subtracted from the current deadline of all the subtransactions that use the task.*
7. *If the current deadline of any transaction is negative then the algorithm is not able to find a feasible solution. If the current deadline of all the transactions is positive go to (8).*
8. *Go to (3) until all the tasks are scheduled.*

In the following example, this algorithm is applied to the problem instance shown in Figure 5.1; the different steps of the algorithm are as shown below.

1. *The complex transaction t_2 consists of two subtransactions $[T1, T3, T5, T10]$ and $[T1, T6, T7, T10]$.*
2. *The end-to-end deadline for each of these subtransactions is the same as the end-to-end deadline for the complex transaction, which is equal to 8.*
3. *Processor P_5 is selected.*
4. *Using EQF, the local deadline for task T_6 is 2 and for task T_8 is 3.6.*
5. *Using fixed-priority scheduling, the worst-case response times for tasks T_6 and T_8 are 1 and 4, respectively.*
6. *The current deadline for transaction t_3 is 2 and for subtransaction $[T1, T6, T7, T10]$ is 7.*
3. *The next processor selected for scheduling is P_6 .*
4. *The local deadlines for tasks T_7 and T_9 are $7/3$ and 2. Task T_9 has the smaller deadline and consequently the higher priority.*
5. *The worst-case response times for the tasks T_7 and T_9 are 2 and 3, respectively.*
6. *The current deadline for the transaction t_3 is 0 and for the subtransaction $[T1, T6, T7, T10]$ is 4.*
3. *Processor P_2 is scheduled next.*
4. *The local deadlines for tasks T_2 and T_3 are 3 and 2, respectively.*
5. *The worst-case response time for T_3 is 1 and for T_2 is 3.*

6. *The current deadline for transaction $t1$ is 3 and for subtransaction $[T1, T5, T3, T10]$ is 7.*
3. *The next processor selected for scheduling is $P3$.*
4. *The local deadlines for tasks $T4$ and $T5$ are 3 and $7/3$, respectively. Task $T5$ has the smaller local deadline and consequently higher priority.*
5. *The worst-case response time for $T4$ is 3 and for $T5$ is 1.*
6. *The current deadline for transaction $t1$ is 0 and for subtransaction $[T1, T5, T3, T10]$ is 6.*
3. *Processor $P1$ is scheduled next.*
4. *The local deadline for task $T1$ is not important because this is the only task running on $P1$.*
5. *The worst-case response time of $T1$ is same as its worst-case execution time and is equal to 1.*
6. *The current deadline for both subtransactions are affected. The current deadline of subtransaction $[T1, T5, T3, T10]$ is reduced from 7 to 5 and subtransaction $[T1, T6, T7, T10]$ is reduced from 6 to 3.*
3. *The last processor is $P4$.*
4. *The local deadline for task $T10$ is not important because this is the only task running on $P4$.*
5. *The worst-case response time for task $T10$ is the same as its worst-case execution time and is equal to 1.*
6. *The current deadlines of both subtransactions are affected. The current deadline of subtransaction $[T1, T5, T3, T10]$ is reduced from 5 to 4 and subtransaction $[T1, T6, T7, T10]$ is reduced from 3 to 2.*

The current deadline for all the transactions and subtransactions is positive, therefore a feasible priority assignment for the problem instance has been found.

5.4 Extending WP to more Complex Problem Domains

The extension of WP to handle revisiting transactions and transactions with subtransactions is described. In the case of revisiting transactions, the WP algorithm (as described in Chapter 4) cannot be used to assign tasks to the worst-case positions. For revisiting transactions, as opposed to the flow shop, the worst-case response times of the tasks are not identical at the current worst-case position. In the extended algorithm a solution is introduced. The algorithm is also extended to solve transactions with subtransactions. WP

needs to be extended in order to (1) map the complex transaction end-to-end deadlines to the subtransactions end-to-end deadlines and, (2) handle tasks that are shared by different subtransactions.

5.4.1 Revisiting Transactions

In the case of revisiting transactions, WP cannot be used in the same way as described in Chapter 4. Unlike in Chapter 4, the highest response time (HRT) at current worst-case position is not the same as a task's worst-case response time. In the case of revisiting, HRT provides a measure for determining which cell can result in the highest response time for a task. HRT is obtained by solving Equation 5.1 on each processor j and choosing the largest value from the set of all R 's produced. In this equation $unschedule(j)$ is the set of all the unscheduled tasks on processor j , and C and Q represent a task's execution time and period respectively.

$$R^{n+1} = \sum_{\forall k \in unschedule(j)} \left\lceil \frac{R^n}{Q_k} \right\rceil C_k \quad (\text{EQ 5.1})$$

For example, in Figure 5.2, initially the current worst-case position corresponds to the lowest priority position on processor $P2$. At the current worst-case position, the worst-case response time for task $T1$ is 8, for task $T2$ is 6, and for task $T3$ is 9; the corresponding HRT is 14. This shows that HRT is not the same as a task's worst-case response time.

In these cases, for determining the worst-case position, the same procedure as introduced in Chapter 4 is followed but the flexibility of transactions is computed differently. Equation 5.2 is used for computing the flexibility of transactions when different tasks T_k are assigned to the worst-case position. In this equation, *DEADLINE* corresponds to the current deadline of the transaction containing the task assigned to the worst-case position. *WCRT* refers to the worst-case response time of the task, and *WORK* refers to the remain-

ing execution time of the transaction. $PR(T_k)$ is the execution time of the current task which is assigned to the worst-case position.

$$FLEXIBILITY = \frac{DEADLINE - WCRT - (WORK - PR(T_k))}{WORK - PR(T_k)} \quad (EQ 5.2)$$

As an example, the algorithm is applied to the problem instance shown in Figure 5.2. The different steps of the algorithm are described below. Note the steps correspond to the algorithm introduced in Chapter 4.

1. *The worst position corresponds to the lowest priority position on P2; the HRT is 14.*
2. *Task T3 cannot be allocated to this position because the corresponding transaction will miss its deadline. For task T2 the WCRT is 6 and by placing T2 at this position the flexibility of t1 becomes equal to $(20-6-9)/9=0.56$. For task T1, WCRT is 8 and by placing T1 at this position the flexibility of t1 becomes equal to $(20-8-7)/7=0.71$. Because of the higher flexibility, task T1 is chosen for this position.*
3. *The current deadline for t1 is $(20-8)=12$.*
1. *The next worst position is the middle priority level on P2; the HRT is 6.*
2. *For this position the WCRT for both tasks T2 and T3 is the same and equal to 6. By putting task T3 at this position the flexibility of transaction t2 is infinity. Task T3 is selected for this position.*
3. *The current deadline for t2 is $(8-6)=2$.*

The algorithm is able to find a feasible solution: on processor P2, task T1 has the lowest priority and task T2 has the highest priority.

5.4.2 Transactions with subtransactions

In this section an algorithm to extend WP to solve transactions with subtransactions is introduced. The algorithm considers all the subtransactions of complex transactions as simple transactions with the same end-to-end deadline as complex transactions. When a task is assigned to the current worst-case position, the resulting flexibility is computed for all of the simple transactions which contain the task. The simple transaction with the low-

est flexibility determines the effect of assigning the task to the current worst-case position. The extended algorithm is described below.

1. *Determine all of the subtransactions of each complex transaction.*
2. *Assign the end-to-end deadline of each complex transaction to all of its subtransactions.*
3. *Find the next cell with the highest response time, assume this cell is located on processor j and the highest response time is equal to HRT .*
4. *For every task T_k that belongs to the set of unscheduled tasks on processor j , assign the task to the cell selected in (3) and find its transaction flexibility as described below.*

If the task belongs to a complex transaction, assign the task to the cell selected in (3) and find the flexibility of each of the subtransactions containing the task using the following equation. In this equation DEADLINE is the current deadline of the subtransaction and WORK is the remaining execution time for the subtransaction. The transaction flexibility is the same as the flexibility of the subtransaction with minimum flexibility.

$$FLEXIBILITY = \frac{DEADLINE - HRT - (WORK - PR(T_k))}{WORK - PR(T_k)} \quad (EQ\ 5.3)$$

If the task does not belong to a transaction with subtransactions then assign this task to the cell selected in (3) and find the transaction flexibility using the above equation.

5. *Select the task which resulted in the highest flexibility (task T_j) and assign it to the cell found in step (3). If the flexibility is negative for all the tasks, there is no feasible solution and the algorithm terminates.*
6. *Current deadlines of the transaction or subtransactions corresponding to the task selected in (5) are changed to $DEADLINE - HRT$. The remaining work for the transaction or subtransactions corresponding to the task are changed to $WORK - PR(T_j)$, where $PR(T_j)$ is the execution time of task T_j .*
7. *If all the tasks on all the processors are not scheduled go to step (3) else terminate.*

In the following example, the above algorithm is applied to the problem instance shown in Figure 5.1. The step numbers correspond to the steps of the algorithm.

1. *The complex transaction $t2$ consists of two subtransactions $[T1, T3, T5, T10]$ and $[T1, T6, T7, T10]$.*
2. *The end-to-end deadline of transaction $t2$ is 8 and is assigned to both subtransactions.*

3. The next cell with the highest response time corresponds to the lowest priority position on processor P5 and the HRT is 4.
4. If task T8 is assigned to this position, the flexibility would be 0. On the other hand, if task T6 is assigned to this position the flexibility will be $1/3=0.33$.
5. Task T6 is assigned to lowest priority position on processor P5, because the flexibility is higher for this task.
6. The current deadline for subtransaction [T1,T6,T7,T10] will be equal to 4. The remaining work for this subtransaction is 3.
3. The next cell with the highest response time corresponds to lowest priority position on either of the processors P6, P2, or P3 and also highest priority position on P5. Processor P5 is selected arbitrarily; the HRT is equal to 3.
4. The only unscheduled task on P5 is T8. So it is assigned to highest priority position on P5.
5. The end-to-end deadline for transaction t3 is 3. The remaining work for t3 is 2.
3. The next cell with the highest response time corresponds to lowest priority position on either of processor P6, P3, or P2. Processor P6 is selected; the HRT is 3.
4. If task T9 is assigned to this position, the flexibility becomes equal to 0/0. So task T9 is assigned to this position.
5. The current deadline for transaction t3 is 0. The remaining work for the transaction is also 0.
3. The next cell with the highest response time corresponds to the lowest priority position on processor P3 or P2. Processor P3 is selected; the HRT is 3.
4. If task T5 is assigned to this position the flexibility will become $2/3$. On the other hand if task T4 is assigned the flexibility will be $1/2$. Task T5 is selected for this position.
5. The current deadline of subtransaction [T1,T3,T5,T10] is 5 and the remaining work is equal to 3.
3. The next cell with highest response time corresponds to the lowest priority position on processor P2; the HRT is 3.
4. If task T3 is selected the flexibility will be 0. On the other hand, if task T2 is assigned to this position the flexibility will be $1/2$. Task T2 is selected for this position.
5. The current deadline of t1 becomes equal to 3 and the remaining work is 2.
3. The next cell with highest response time corresponds to highest priority position on P3; the HRT is 2.
4. The only unscheduled task on P3 is T4. This task is assigned to this position.
5. The current deadline for t1 is 1. The remaining work is 0.
3. The next cell with the highest response time corresponds to either of processors P1, P2, P6, or P4. Processor P1 is selected first; the HRT is 1.

4. *T1 is the only task on P1.*
5. *The end-to-end deadline for both subtransactions will change. The end-to-end deadline for subtransaction [T1,T6,T7,T10] is 3 and for subtransaction [T1,T3,T5,T10] is 4. The remaining work for both subtransactions is 2.*
3. *The next cell with highest response time corresponds to either of processors P2, P6 or P4. Processor P2 is selected; the HRT is 1.*
4. *The only unscheduled task on P2 is T3.*
5. *The current deadline for subtransaction [T1,T3,T5,T10] is 2. The remaining work for the subtransaction is 1.*
3. *The next cell with highest response time corresponds to either P6 or P4. Processor P6 is selected; the HRT is 1.*
4. *The only unscheduled task on this processor is T7.*
5. *The current deadline for subtransaction [T1,T6,T7,T10] is 3. The remaining work for the subtransaction is 1.*
3. *The last cell is on processor P4; the HRT is 1.*
4. *The only unscheduled task on P4 is T10.*
5. *The current deadline for both subtransactions is changed. The current deadline for subtransaction [T1,T3,T5,T10] is 1 and for subtransaction [T1,T6,T7,T10] is 2. The remaining work for both subtransaction is 0.*

So the algorithm is successful in finding a feasible solution for the problem instance shown in Figure 5.1.

5.5 Simulation Model

To study the performance of the heuristic algorithms described in the previous sections, two simulation models (one for transactions with subtransactions and one for revisiting transactions) are proposed. The simulation models assume modest architectures so as to ease the comprehension of experimental results.

5.5.1 Simulation Model for Transactions with Subtransactions

The simulation model uses the architecture shown in Figure 5.3 with the following specifications.

Processors: There are $K+2$ processors in the system. Each processor services its tasks using a fixed priority scheduling algorithm. In Figure 5.3 nodes are organized in 3 stages such that the first and last stage consist of one processor and the middle stage consists of K processors. The middle stage is called the *parallel stage*.

Complex Transactions: A typical complex transaction is shown in Figure 5.3. A complex transaction is assumed to consist of K subtransactions $[T_1, T_2, T_{(K+2)}]$, $[T_1, T_3, T_{(K+2)}]$, , $[T_1, T_{(K+1)}, T_{(K+2)}]$ such that tasks $T_2, \dots, T_{(K+1)}$ are executed in parallel at K different processors. The end-to-end deadline of a complex transaction is calculated using the following formula. The period of a transaction is the same as its end-to-end deadline.

$$dl(t_i) = (pr(T_1) + \max \{pr(T_2), \dots, pr(T_{(K+1)})\} + pr(T_{(K+2)}) \times (1 + fl(t_i))) \quad (\text{EQ 5.4})$$

In this equation $fl(t_i)$ refers to flexibility of the complex transaction t_i , $pr(T_i)$ refers to the execution time of task T_i and $\max \{pr(T_2), \dots, pr(T_{(K+1)})\}$ refers to the execution time of the longest task in the parallel stage. The flexibility of a transaction represents the tightness of its timing constraint (i.e., the more flexible a transaction, the less stringent is its timing constraint). For $fl(t_i)$, a Gamma distribution with a mean of MF and a coefficient of varia-

tion of CVF is assumed. A task's execution time has a Gamma distribution with a mean of ME and coefficient of variation of CVE. The goal of the simulation is to study the effect of the number of processors (K) in the parallel stage on the relative performance of EEQF, EWP and ERTC.

Figure 5.3 Simulation Model for Transactions with Subtransactions

The set of experiments designed to investigate the effect of parameter K is shown in the Table 5.1; each row in the table represents an experiment. In all the experiments the parameter K is varied from 1 to 6 under different settings of CVE and CVF. In all the experiments number of stages (NS) is 3 and number of transactions (NT) is 5. The MF is set at 2.5. This set of experiments is chosen to represent the behavior of algorithms (i.e., the relative performance of algorithms) after analysing the results of an extensive set of experimentation. In the extensive experimentation all the parameters except NS were changed; CVE is varied from 0 to 2.0, CVF from 0 to 1.0, NT from 5 to 10, and MF from 2.5 to 5.0. The effect of ME is studied and it is observed that it has only scaling affect and it does not change the structure of problem instances.

K IS VARIED FROM 1 TO 6	0.25	0.00	1.0	2.5	5	3
	0.25	0.50	1.0	2.5	5	3
	1.00	0.00	1.0	2.5	5	3
	1.00	0.50	1.0	2.5	5	3
	2.00	0.00	1.0	2.5	5	3
	2.00	0.50	1.0	2.5	5	3

Table 5.1 Experiments for Transactions with Subtransactions

5.5.2 Simulation Model for Revisiting Transactions

The simulation model for the revisiting case consists of a number of processors (NP) and a number of transactions (NT). All the transactions consist of the same number of tasks (NJ). The number of tasks can be smaller, equal or greater than the number of processors.

Tasks are allocated to processors as follow. For a given transaction, its constituent tasks are randomly assigned to processors as long as two consecutive tasks of the transaction are not allocated to the same processor. For example, if tasks $T_1, \dots, T_r$ are assigned to processors then the next task $T_{(r+1)}$ can be allocated to any of the processors except the one to which T_r (its neighbor task) is allocated.

End-to-end deadlines of transactions are computed using Equation 5.5. The periods of transactions are the same as their end-to-end deadlines. In Equation 5.5, $fl(t)$ is the flexibility of transaction t and it represents the tightness of the transaction's timing constraint; $pr(T_j)$ is the execution time for task T_j . In this equation, NJ is the number of tasks in a transaction.

$$dl(t) = (1 + fl(t)) \times \sum_{j=1}^{NJ} pr(T_j) \quad (\text{EQ 5.5})$$

For modeling the flexibility of transactions a Gamma distribution with mean of MF and coefficient of variation of CVF is used. The task execution times have a Gamma distribution with a mean of ME and a coefficient of variation of CVE.

The goal of the simulation model is to investigate the effect of revisiting on the relative performance of the EEQF and EWP. The effect of the revisiting is investigated through the number of tasks (NJ). For a given number of processors, by increasing NJ, the number of revisiting tasks in a problem instance is increased.

To limit the experiment space, we have decided to keep the number of processors at 4. The set of experiments designed to investigate the effect of revisiting is shown in the Table 5.2; each row represents an experiment. In all these experiments, NJ is varied under different settings of CVE, CVF and MF. When MF is 3.0, NJ is changed from 3 to 8 and when it is equal to 2.0, NJ is varied from 3 to 5. When MF is 2.0, increasing number of tasks beyond 5 results in a substantial drop in the number of solutions found by either of the algorithms. This set of experiments is chosen to represent the relative performance of EEQF and EWP after analysing the results of a higher number of experiments. In the full experimentation the CVE is varied from 0 to 2.0, CVF from 0 to 0.5, NT from 5 to 10, and MF from 2.0 to 6.0. The ME is not changed since it has only scaling affect and it does not change the structure of problem instances.

Table 5.2 Experiments For The Revisiting

5.6 Simulation Results

Simulations results in this section show the effect of subtransactions and revisiting transactions on the relative performance of the extended algorithms. The extended algorithms are developed in the C language and they are tested by tracing them over a number of problem instances.

5.6.1 Transactions with subtransactions

Figures 5.4 to 5.9 show the relative performance of EEQF and EWP under different settings of CVE and CVF when parameter K is varied from 1 to 6.

In Figures 5.4 and 5.5, CVE is 0.25, and CVF is set at 0 and 0.5 respectively. In both Figures, by increasing the parameter K the number of solutions found by only EWP is increased. This phenomena can be explained as follows. The probability of failure of a complex transaction to meet its end-to-end deadline is affected by both the number of its subtransactions and also the probability of failure of each subtransaction to meet the end-

to-end deadline. The greater the number of subtransactions and higher the probability of missing the end-to-end deadline, the higher will be the probability that a complex transaction fails to satisfy its end-to-end deadline. EWP and EEQF are extensions of the WP and EQF algorithms and it is shown in Chapter 4 (for a flow shop) that WP can find solutions in many cases for which EQF is not successful. By increasing number of subtransactions (increasing parameter K) it is more likely that EEQF fails for one of the subtransactions for which EWP can find a feasible solution.

In Figure 5.6, CVE is increased to 1.0. By increasing CVE, the problem instances become less challenging and both algorithms are successful in finding feasible solutions under different values of K .

In Figure 5.7, keeping CVE at 1.0, CVF is increased to 0.5. By increasing K the number of solutions found by only EWP tends to decline. By increasing CVE from 0.25 to 1.0 a change in the relative performance of EEQF and EWP is observed (i.e., when CVE is 0.25 by increasing K more solutions are found by only EWP and when CVE is 1.0 by increasing K less solutions are found by only EWP); this phenomena is explained as follow. Every complex transaction consists of K subtransactions and the subtransaction with the highest task execution time in the parallel stage determines the flexibility of all K subtransactions. Thus by increasing K , the utilization on all the processors drops (especially at higher value of CVE) and consequently the probability of failure of a subtransaction to meet its end-to-end deadline drops. The higher is the CVE, the higher is the effect of increasing K in reducing the probability of failure of a subtransaction. Thus, it can be concluded that at a high value of CVE, increasing K drops the probability of failure of subtransactions to a level that EEQF is able to find more solutions as K increases.

In Figures 5.8 and 5.9, CVE is increased to 2.0. By increasing CVE to 2.0 problem instances become even less challenging and both algorithms are able to solve problem instances.

Figures 5.10 to 5.15 show the relative performance of ERTC and EWP under different settings of CVE and CVF when parameter K is changed from 1 to 6.

All the experiments show that the performance of ERTC improves by increasing K . This can be observed even at low value of CVE (Figures 5.10 and 5.11). By increasing K , processor utilization drops which reduces task minimum response times and consequently increases subtransaction remaining slack times. These experiments show that ERTC is quite sensitive to processor utilization.

In Figures 5.12 to 5.15, by increasing CVE to higher values, problem instances become less challenging and both algorithms EWP and ERTC are able to solve problem instances.

Experimental results in this section show that by increasing K the probability of failure of a subtransaction drops (more pronounced at higher values of CVE). At low values of CVE, though, increasing K reduces the probability of failure of a subtransaction but still as K increases a higher proportion of solutions are solved by only EWP (when compared to EEQF). By increasing CVE, the effect of increasing K on reducing the probability of failure of a subtransaction becomes stronger and the performance of algorithms gets closer as K increases. For ERTC even at low values of CVE, as K increases the performance of ERTC improves.

Figure 5.4 The Effect of Subtransactions for EEQF and EWP (CVE=-0.25 CVF=0.0)

Figure 5.5 The Effect of Subtransactions for EEQF and EWP (CVE=-0.25 CVF=0.5)

Figure 5.6 The Effect of Subtransactions for EEQF and EWP (CVE=1.0 CVF=0.0)

Figure 5.7 The Effect of Subtransactions for EEQF and EWP (CVE=1.0 CVF=0.5)

Figure 5.8 The Effect of Subtransactions for EEQF and EWP (CVE=2.0 CVF=0.0)

Figure 5.9 The Effect of Subtransactions for EEQF and EWP (CVE=2.0 CVF=0.5)

Figure 5.10 The Effect of Subtransactions for ERTC and EWP (CVE=-0.25 CVF=0.0)

Figure 5.11 The Effect of Subtransactions for ERTC and EWP (CVE=-0.25 CVF=0.5)

Figure 5.12 The Effect of Subtransactions for ERTC and EWP (CVE=1.0 CVF=0.0)

Figure 5.13 The Effect of Subtransactions for ERTC and EWP (CVE=1.0 CVF=0.5)

Figure 5.14 The Effect of Subtransactions for ERTC and EWP (CVE=2.0 CVF=0.0)

Figure 5.15 The Effect of Subtransactions for ERTC and EWP (CVE=2.0 CVF=0.5)

5.6.2 Revisiting Transactions

Figures 5.16, 5.17, 5.18, and 5.19 show the relative performance of EEQF and EWP for different settings of CVE and CVF. In all the figures MF is 3.0 and the number of tasks is varied from 3 to 8. The numbers on the bars show the average ratio of revisiting tasks per problem instance. The average ratio of revisiting tasks is the ratio of the number of revisiting tasks to the total number of tasks in a problem instance. In all the figures, as NJ is increased, the average ratio of revisiting tasks is also increased (the ratio is increased from 0.11 to 0.52).

Figure 5.16 shows the relative performance of EEQF and EWP when CVE is 0.5, and CVF is 0.0. Initially both algorithms are able to find feasible solutions but by increasing NJ and consequently increasing the average ratio of revisiting tasks, many feasible solutions are only found by EWP. When NJ is 8, close to 50% of the solutions are found by only EWP. In Figure 5.17, CVF is increased to 0.5. Initially (when NJ is 3, 4 and 5) the number of solutions found by only EWP exceed those found when CVF is 0.0. This phenomena does not continue for higher NJ. For higher NJ the problem instances become more difficult and many problem instances are discarded. By rejecting difficult problem instances, the statistical distribution of problem instances is affected and algorithms are compared on simpler problems. This explains why many solutions are not found by only EWP in Figure 5.6 when NJ is high.

Figure 5.18 shows the relative performance when CVE is 1.0, and CVF is 0.0. Both algorithms are able to find solutions when NJ is varied from 3 to 8. Increasing CVE from 0.5 to 1.0 makes the problem instances less challenging and both algorithms are able to find solutions. In Figure 5.19, the CVF is increased to 0.5. Figure shows that as NJ is increased more solutions are found by only EWP.

To confirm that EWP performs better than EEQF an experiment is designed as follows. For each problem instance the total number of revisiting tasks is computed. For NP equal to 4, NT equal to 5, and NJ equal to 7, the minimum number of revisiting tasks in a problem instance is 15 and the maximum is 25. Problem instances are classified based on their number of revisiting tasks. In this case problem instances belong to one of the 11 bins. If the total number of revisiting tasks is 15 the problem instance belongs to the first bin and if the total number of revisiting tasks is 25 the problem instance belongs to the last bin. In the experiment 50000 problem instances are generated and for each problem instance the number of revisiting tasks is computed. The number of revisiting tasks determines to which of 15 bins the problem instance belongs. The histogram in Table 5.2 shows the result.

In Table 5.2, the column *ratio* represents the ratio of the solution found by only EWP to the solutions found by either of algorithm for a given bin. For example, from 50000 generated problem instances, 11120 of them have 15 revisiting tasks. Applying EEQF and EWP to these problem instances, solutions for 85 of the problem instances are found by only EEQF, solutions for 1074 are found by only EWP and solutions for 9634 are found by both algorithms. In total, in bin 15, 10793 solutions are found by either algorithm of which 9.9% are found by only EWP. Table 5.2 shows that the *ratio* is higher for problem instances which have a higher number of revisiting tasks. This confirms that when more revisiting tasks exist in a problem instance, EWP shows a better performance in finding a feasible solution. In Table 5.2, a drop in *ratio* for bin 20, and 21 is observed. The reason for this unexpected drop, is a high rejection of problem instances by both algorithms in these bins. Introducing more revisiting tasks make the problem instances more difficult and a higher percentage of problem instances are discarded by both algorithms. This has resulted in an insufficient sample size for determining a valid ratio.

Figure 5.16 The Effect of Revisiting (CVE=0.5 CVF=0.0 MF=3.0)

Figure 5.17 The Effect of Revisiting (CVE=0.5 CVF=0.5 MF=3.0)

Figure 5.18 The Effect of Revisiting (CVE=1.0 CVF=0.0 MF=3.0)

Figure 5.19 The Effect of Revisiting (CVE=1.0 CVF=0.5 MF=3.0)

Total Revisits (Bin #)	Number of Problem Instances in this Bin	Solutions Found by only EEQF	Solutions Found by only EWP	Solutions Found by both Alg.	Ratio
15	11120	85	1074	9634	0.099
16	19099	108	3122	14977	0.171
17	13682	76	2891	9814	0.226
18	4960	19	1227	3293	0.270
19	1028	3	277	615	0.309
20	108	1	23	70	0.244
21	3	0	0	2	0.00
22	0	0	0	0	----
23	0	0	0	0	----
24	0	0	0	0	----
25	0	0	0	0	----

**Table 5.3 Distribution of Problem Instances when CVE=0.5, CVF=0.0, MF=3.0
NT=5, NP=4, and NJ=7**

Figures 5.20, 5.21, 5.22, and 5.23 show the results of experiments when MF is 2.0. In all these figures, the result is only reported for NJ equal to 3,4 and 5. For higher NJ most of problem instances are discarded since neither algorithm is able to find a feasible solution. In all the figures, as NJ is increased the average ratio of revisiting is increased from 0.11 to 0.32.

In Figure 5.20, CVE is 0.5 and CVF is 0.0. As NJ is increased from 3 to 5, a higher percentage of solutions is found by only EWP. When NJ is 5, close to 45% of the solutions are found by only EWP. By increasing CVF to 0.5 (Figure 5.21), initially (when NJ is 3), the number of solutions found by only EWP shows an increase when compared to Figure 5.20. For higher NJ the number of solutions found by only EWP either remained constant (NJ equal to 4) or dropped (NJ equal to 5). For higher NJ, problem instances become more difficult and more of them are discarded since neither algorithm is able to find a solu-

tion. Discarding difficult problem instances is the reason for the drop in the number of solutions found by only EWP for higher NJ.

In Figure 5.22, CVE is 1.0 and CVF is 0.0. Comparing to Figure 5.20, the only parameter that is changed is CVE. The value of CVE is increased from 0.5 to 1.0. By increasing the value of CVE the problem instances become less challenging and consequently the performance of the algorithms gets closer. In Figure 5.23, CVF is increased to 0.5. In this Figure as NJ is increased more solutions are found by only EWP.

Table 5.3 shows the distribution of 50000 generated problem instances into different bins when CVE is 0.5, CVF is 0.0, MF is 2.0, and NJ is 4. The only differences between this experiment and the results reported in Table 5.2 is the value of MF which is decreased to 2.0 and the value of NJ which is decreased from 7 to 4. For 4 processors and 4 tasks, the minimum number of revisiting tasks in a problem instance is 0 and the maximum number of revisiting tasks is 10. In Table 5.3, it is observed that as bin number increases the *ratio* also increases. This experiment also confirms that for problem instances that have more revisiting tasks, EWP is able to find more solutions than EEQF. In Table 5.3, the value of *ratio* drops for bin number 9 and 10 which is the result of too few problem instances in these bins.

Figure 5.20 The Effect of Revisiting (CVE=0.5 CVF=0.0 MF=2.0)

Figure 5.21 The Effect of Revisiting (CVE=0.5 CVF=0.5 MF=2.0)

Figure 5.22 The Effect of Revisiting (CVE=1.0 CVF=0.0 MF=2.0)

Figure 5.23 The Effect of Revisiting (CVE=1.0 CVF=0.5 MF=2.0)

Total Revisits (Bin #)	Number of Problem Instances in this Bin	Solutions Found by only EEQF	Solutions Found by only EWP	Solutions Found by both Alg.	Ratio
0	29	0	0	29	0.00
1	398	1	19	377	0.047
2	2517	15	184	2307	0.073
3	8242	57	1006	7013	0.124
4	14568	110	2251	11571	0.161
5	14492	102	2706	10661	0.200
6	7391	41	1611	5034	0.240
7	2003	10	491	1246	0.281
8	324	1	95	186	0.336
9	34	0	5	22	0.185
10	2	0	0	2	0.00

Table 5.4 Distribution of Problem Instances When CVE=0.5, CVF=0.0, MF=2.0, NT=5, NP=4, and NJ=4

Looking at the results of experiments in this section, it can be concluded that EWP is able to find more solutions compared to EEQF when revisiting tasks exist in the system. The reason is explained as follows. In each iteration of the EEQF algorithm, an unscheduled processor is selected and priorities are assigned to all the tasks running on that processor. Using the assigned priorities, the worst-case response times of tasks are computed and is subtracted from current deadline of each of the transactions. The initial deadline of a transaction is the same as its end-to-end deadline. If several revisiting tasks belonging to one transaction exist on a processor, EEQF will assign priorities to all these revisiting tasks in one step and it is very likely that the transaction misses its deadline. However, in EWP, the priority assignment for the revisiting tasks of a transaction can be mixed with the priority assignment for the revisiting tasks of other transactions and consequently it is more likely that the algorithm will find a feasible solution.

5.7 Complexity of Algorithms

The computational complexities of EQF, WP, RTC, EEQF, EWP, and ERTC are discussed. Two components contribute to the overall computation time required by any of these algorithms. The first component corresponds to the computation needed for the priority assignment and the second corresponds to the computation of the worst-case response time for each task.

The priority assignment computation for EQF and EEQF (revisiting case) is the calculation of task local deadlines, for WP and EWP (revisiting case) is the calculation of transaction flexibilities, and for RTC is the division of the remaining slack time for each transaction among its tasks. The second component is the computation of task worst-case response time which is calculated using either of Equation 3.2 or Equation 3.10 (Equation 3.10 is used for revisiting case); the complexity of solving these equations is dependent on workload parameters. Consider for example the computation of R_i , the worst-case response time of task T_i , by Equation 3.2. In the worst-case scenario the number of iterations of Equation 3.2 to compute R_i is $\sum_{\forall k|(k \in hp(i))} \lceil R_i/Q_i \rceil$. Since R_i can not be greater than Q_i , the number of iterations needed to calculate the response time of task T_i is upper bounded by $\sum_{\forall k|(k \in hp(i))} \lceil Q_i/Q_k \rceil$ which shows the number of iterations depends on the ratio of the period of task T_i to the period of tasks that have higher priority than T_i . So, the complexity of computing a task worst-case response time depends on the relative periods of tasks allocated to a processor.

The task worst-case response time computation is selected as a unit of computation for determining the overall complexity of the algorithms because: (1) it is invoked by all algorithms introduced in this research and by other researchers (for example in HOPA), and (2) it is the dominant computation (because it involves the solution of a recurrence relation) in all the algorithms. Thus, the complexity of the algorithms is determined by the

number of times Equation 3.2 or 3.10 is solved for computing task worst-case response times. For a system consisting of N tasks, the number of times Equation 3.2 or Equation 3.10 is called is of the order of N for EQF, WP, RTC, and EEQF (revisiting case) and is of the order of N^2 for EWP (revisiting case). For EWP (revisiting case), the task's worst-case response time is not the same for all tasks at the current worst-case position. Thus, the worst-case response time must be calculated for each task separately and consequently the complexity of the algorithm is of the order of N^2 . In the case of transactions with subtransactions, the complexity of algorithms is dependent on the software architecture of the complex transactions. The priority assignment computation for EEQF is the calculation of task local deadlines, for EWP is the calculation of complex transactions flexibilities, and for ERTC is the division of subtransactions remaining slack times among tasks. For these algorithms the task worst-case response time computation is not necessarily the dominant computation. Since a complex transaction with many forks may exist in the system, each of these algorithms must analyse several subtransactions in order to assign priority to tasks. Therefore for certain problem instances the computational complexity of the priority assignment component may be higher than the computational complexity of the worst-case response time computation for tasks.

Table 5.5 shows the CPU time consumed when the algorithms are compared on a SPARC 2 workstation. Each pair of the algorithms are compared over 100 problem instances. The results show that for a considerable number of tasks ($N=50$) allocated to 5 processors in the system, algorithms are able to find feasible solutions in a reasonable amount of time. In the case of revisiting and transactions with subtransactions, the models defined in section 5.5 and 5.6 are used to compare the algorithms. The model used for flow shop is the same as the one described in section 4.3.

Algorithm pair	Model	NT	NP	NS	K	CPU Time
[EQF, WP]	Flow Shop	10	5	-	-	7 sec
[EQF, RTC]	Flow Shop	10	5	-	-	7 sec
[EEQF, EWP]	Revisiting	10	5	-	-	60 sec
[EEQF, EWP]	Transactions with Subtransactions	10	5	3	3	8 sec
[EEQF, ERTC]		10	5	3	3	8 sec

Table 5.5 CPU Time for Comparing the Algorithms

5.8 Case Study

The towed array signal processing example from Chapter 1 (shown again here in Figure 5.24) is used as a case study for the extended algorithms. The towed array sonar uses a cable with an array of N hydrophones (undersea microphones) to passively detect sounds underwater. It is towed behind a ship or submarine. Signal processing is needed to combine the many different signal sources to produce 360 degrees of conical beams around the array. The example uses only transactions with subtransactions, thus the revisiting algorithms are not employed. The basic system configuration shown in Figure 5.24 has the nominal processing times and periods given in Table 5.5. There are 12 transactions in the basic configuration; these transactions are described below.

1. Parameter Control Transactions (t_7 , t_8 , t_9 , t_{10} , t_{11} , and t_{12})

These transactions are used to change signal processing parameters, such as the resolution of the spectral analysis.

2. Retrieval Transactions (t_2 , t_3 , t_4 , and t_5)

Transactions t_2 through t_5 retrieve results distributed over 4 processors (three are the sonar data time-frequency-intensity or "lofargram" plots for various types of processing, and one is for tracking data).

3. Raw Data Transaction (tl)

This transaction represents raw sensor data that enters the system and initiates a variety of signal processing functions. The raw data is first passed to each of the three beamforming processes. Each beamforming process takes the data from the N hydrophones, performs an FFT (Fast Fourier Transform) to create frequency bins, performs phase shifting of the data values, and then a summation to form a group of K conical beams. The three set of beams collectively represent 360 degrees of coverage around the towed array. The audio task takes the output of the beamforming process and converts the data into an audio format for the headphones. The output from a beamforming process is passed to the peer narrowband process where each beam goes through spectral analysis to determine the power level of the signal in frequency bins at some specified resolution. The result at the end of processing is a spectral power in a set of frequency bin that can be displayed as a pixel intensity on a frequency-time plot, or lofargram (frequency is the x-axis, time is the y-axis). The BBDE task is also triggered after the beamforming is completed. It takes all of the beams and for each beam, computes the average power in the beam over all the beam frequencies (the broadband power). The resulting power can be displayed as a pixel intensity on a beam-time plot (the beam number is the x-axis, and time is the y-axis). It also performs a spectral analysis of the time series of broadband power values for each beam (demodulated noise or Demon analysis). This shows the frequencies in the change of the broadband power over time. The spectral power is displayed in a frequency-time plot like the narrowband results. The tracking task is triggered at the end of narrowband processing. It examines the beam data of some specified set of beams over a narrow set of frequencies to follow automatic or operator initiated tracks. It plots the track on a beam-time diagram, with fixed intensity. It also computes the beam with the highest power in the specified beam frequency neighborhood. The tracking uses this data to estimate the direction of the object being tracked.

4. Tracking Parameter Change (t6)

This transaction alters the parameters used in tracking, such as the number of beams and frequency neighborhood. It is also used to specify the operator initiated tracks.

Figure 5.24 Transactions inside Signal Processing of a Sonar System

Task Name	Task Execution Time	Task Corresponding Transaction	Transaction's period	Transaction's Deadline
RDC	130(ms)	t_1	750(ms)	750(ms)
B1	90(ms)	t_1	750(ms)	750(ms)
B2	90(ms)	t_1	750(ms)	750(ms)
B3	90(ms)	t_1	750(ms)	750(ms)
A1	50(ms)	t_1	750(ms)	750(ms)
A2	50(ms)	t_1	750(ms)	750(ms)
A3	50(ms)	t_1	750(ms)	750(ms)
BBDE	120(ms)	t_1	750(ms)	750(ms)
NB1	210(ms)	t_1	750(ms)	750(ms)
NB2	120(ms)	t_1	750(ms)	750(ms)
NB3	210(ms)	t_1	750(ms)	750(ms)
TC	130(ms)	t_1	750(ms)	750(ms)
DS1	40(ms)	t_2	750(ms)	750(ms)
DS2	40(ms)	t_3	750(ms)	750(ms)
DS3	40(ms)	t_4	750(ms)	750(ms)
DS4	10(ms)	t_5	750(ms)	750(ms)
TP	10(ms)	t_6	10000(ms)	1500(ms)
PC1	10(ms)	t_7	10000(ms)	1500(ms)
PC2	10(ms)	t_8	10000(ms)	1500(ms)
PC3	10(ms)	t_9	10000(ms)	1500(ms)
PC4	10(ms)	t_{10}	10000(ms)	1500(ms)
PC5	10(ms)	t_{11}	10000(ms)	1500(ms)
PC6	10(ms)	t_{12}	10000(ms)	1500(ms)

Table 5.6 Task Execution Times, Periods, and Deadlines for Signal Processing Module of the Towed Array Sonar System

Two analyses are performed using the EEQF, EWP, and ERTC algorithms on this example. In the first analysis, the basic configuration is used. The second analysis assumes that data from two sensor sources is available; i.e., two towed arrays. This situation could exist

if the data from towed arrays of two ships were analysed at a single site, the primary benefit of which is determining the direction of the target (normally a towed array cannot fix the direction of a target as originating from the left or right side of the ship, without actually changing the course of the ship to generate data from different angles). Multiple data sources can also help to provide target range. For the analysis, the data rate from the second sensor is reduced on the assumption that full data rate may not be practical in ship-to-ship transmission.

Analysis 2 uses two copies of the Raw Data transaction; the first as in the basic configuration and the second at 1.5 times the period of the basic configuration, but processing the same set of data values.

The results of applying the algorithms to the above two scenarios are presented in Table 5.6 and 5.7. The tables show task priorities as computed by each of the algorithm for the two scenarios explained above. Table 5.7 shows the task priorities for scenario 2. In this scenario another copy of the raw data transaction exists in the system. Tasks belonging to this transaction are named the same as the original transaction's tasks but with an added * symbol. In Table 5.6, tasks with the symbol * do not have priorities since the second raw data transaction does not exist in the system. Task priorities are shown by integer numbers; the lower the number the higher the priority of a task. Each audio task (e.g., A1) is triggered by its corresponding beamforming task (e.g., B1) which is allocated to the same processor as the audio task. In Chapter 3, the same priority is assumed for tasks belonging to a common transaction and running on the same processor. Following this assumption each audio and its corresponding beamforming task are considered as a single task (e.g., B1A1) and the same priority is assumed for both of them.

For the first scenario all three algorithms are successful in finding a feasible priority assignment and the task's priorities found by each of the algorithm are the same. For the

second scenario, feasible priority assignments are found only by EEQF and EWP and each algorithm found the same task priorities. The algorithm ERTC is not able to find a feasible solution for the second scenario. The reason for the failure of the algorithm is described below.

Starting with transaction t_1 (the algorithm starts with this transaction since the deadline of the transaction is the lowest among all the unscheduled transactions) ERTC computes the minimum response times of tasks RDC, B1A1, NB1 and TC which are 130, 140, 210 and 130 respectively. The corresponding local deadlines for these tasks are 159, 172, 260 and 159. Now, considering subtransaction $[RDC^*, B1A1^*, NB1^*, TC^*]$ which belongs to the other raw data transaction, the minimum response times for these tasks are 260, 280, 420, and 260. Neither of these tasks can get the highest priority on the processor they are allocated to because the scheduled tasks RDC, B1A1, NB1 and TC will miss their local deadlines. The summation of the minimum response times for the tasks RDC^* , $B1A1^*$, $NB1^*$, and TC^* is greater than their corresponding subtransaction end-to-end deadline and consequently the algorithm fails to find a feasible priority assignment.

Task Name	Priority (EEQF)	Priority (EWP)	Priority (ERTC)
RDC	0	0	
B1	0	0	0
B2	0	0	0
B3	0	0	0
A1	0	0	0
A2	0	0	0
A3	0	0	0
BBDE	1	1	1
NB1	0	0	0
NB2	0	0	0
NB3	0	0	0
TC	0	0	0
RDC*	--	--	--
B1*	--	--	--
B2*	--	--	--
B3*	--	--	--
A1*	--	--	--
A2*	--	--	--
A3*	--	--	--
BBDE*	--	--	--
NB1*	--	--	--
NB2*	--	--	--
NB3*	--	--	--
TC*	--	--	--
DS1	1	1	1
DS2	2	2	2
DS3	1	1	1
DS4	1	1	1
TP	2	2	2
PC1	1	1	1
PC2	3	3	3
PC3	2	2	2
PC4	2	2	2
PC5	1	1	1
PC6	1	1	1

Table 5.7 Task's Priorities for Scenario 1

Task Name	Priority (EEQF)	Priority (EWP)	Priority (ERTC)
RDC	1	1	--
B1	0	0	--
B2	0	0	--
B3	0	0	--
A1	0	0	--
A2	0	0	--
A3	0	0	--
BBDE	1	1	--
NB1	0	0	--
NB2	0	0	--
NB3	0	0	--
TC	0	0	--
RDC*	0	1	--
B1*	1	1	--
B2*	1	1	--
B3*	1	1	--
A1*	1	1	--
A2*	1	1	--
A3*	1	1	--
BBDE*	3	3	--
NB1*	1	1	--
NB2*	2	2	--
NB3*	1	1	--
TC*	1	1	--
DS1	2	2	--
DS2	4	4	--
DS3	2	2	--
DS4	2	2	--
TP	3	3	--
PC1	2	2	--
PC2	5	5	--
PC3	3	3	--
PC4	3	3	--
PC5	2	2	--
PC6	1	1	--

Table 5.8 Tasks's Priorities for Scenario 2

5.9 Summary

In this chapter, algorithms EQF, RTC and, WP developed in Chapter 4 are extended in order to solve other problem domains (i.e., revisiting transactions and transactions with subtransactions). Extended algorithms are compared and the following conclusions are made.

- By increasing the number of subtransactions, in overall, EWP is better able to find feasible solution compared to EEQF.
- ERTC shows an improvement in performance as the number of subtransactions increases. However, EWP still finds more solutions by comparison.
- EWP performs better than EEQF when the number of revisiting tasks is increased.

In the last section of the chapter, the extended algorithms were used to find priorities for different tasks in the signal processor module of a towed array sonar system. For this example, feasible solutions were found by both EEQF and EWP.

Chapter 6

Summary and Conclusions

6.1 Summary

This thesis is concerned with the problem of scheduling transactions in hard real-time multiprocessor systems, typical of those found in signal processing and real-time communication applications. In our model, the real-time system is a collection of transactions; a transaction is composed of a set of tasks, each of which executes on one processor. The timing constraints of transactions are end-to-end, meaning that we are not concerned with the finishing time of the tasks as long as the end-to-end timing constraints are met. In this research, meeting end-to-end deadlines are the timing constraints on the transactions. It is assumed that tasks are allocated to the processors and the problem is the scheduling of the tasks such that all transactions can meet their end-to-end deadlines.

A scheduling method is given in Chapter 3. When the workload consists of periodic tasks and fixed priority scheduling is used on processors then the worst-case response times of tasks can be computed on every processor and an upper bound can be calculated for the end-to-end delay of every transaction. Although the transactions are initiated periodically, because of the interactions among tasks on processors, the tasks on intermediate processors can become aperiodic. Traffic reshaping is assumed at the boundary of each processor to guarantee that all tasks are initiated periodically. In this chapter a new upper bound for the end-to-end delay of the revisiting transaction (a revisiting transaction goes through at least one of the processors more than one) is introduced. A substantial amount of time may separate consecutive visits of a revisiting transaction on a processor and it is pessi-

mistic to consider the contribution of all the revisiting tasks for computing the response times of other tasks. An algorithm is presented which takes into account the phasing of revisiting tasks in order to improve the bound on the end-to-end response times of transactions.

To satisfy the end-to-end deadline of a transaction, a feasible priority assignment must be found for the transaction's tasks. It is known that the problem of priority assignment is NP-hard. In Chapter 4, new heuristic algorithms for the flow-shop model are introduced and it is shown through simulation that in many situations they outperform algorithms introduced by other researchers (i.e., they can find feasible solutions in the cases that other heuristics are not successful). Starting with an existing algorithm LM (laxity monotonic) [3], a problem with the algorithm is identified. The LM algorithm assign all of a transaction's slack time to the task on the processor being currently scheduled and it does not reserve a portion of the slack time for the future activities of the transaction. To alleviate this problem a new heuristic algorithm EQF (equal flexibility) is developed. The EQF algorithm gives a portion of the total slack time to the tasks on the processor being scheduled and it saves a portion of the slack time for the future activities of transactions. Through simulation, it is shown that EQF can find solutions for many cases in which LM is not successful, especially for systems with a large number of processors. To overcome some of the problems with EQF another algorithm WP (worst position) is introduced. In each step of EQF priorities are assigned to all the tasks running on a processor. A task may receive a priority such that it consumes a substantial slack time (a lot more than what was considered for it by the EQF algorithm) and does not leave enough slack time for other tasks in the transaction. This phenomena is more pronounced when the load is distributed on the smaller number of processors and processors are highly utilized (note: to satisfy the end-to-end deadline of transactions, processor utilization should be reduced when the number of processors is increased). In WP, this problem is reduced by checking that the

assignment of a priority to a task does not cause a substantial slack time to be taken from the transaction. Through simulation, it is observed that WP can solve many problem instances in which EQF is not successful. It is shown that by increasing the number of processors the performance of EQF and WP gets closer. The performance of WP is also compared to HOPA (Heuristic Algorithm for Optimized Priority Assignment) [12] and it is observed that HOPA cannot find solutions for many problem instances for which WP can find a solution. It is shown that for larger numbers of processors HOPA is less successful in finding feasible solutions. In [17] an algorithm for establishing real-time channels is proposed. The algorithm is modified in this research and is used for assigning priority to tasks. It is shown that modified algorithm (RTC: real-time channel) has a better performance compared to the one proposed in [17]. The RTC algorithm can be used for scheduling dynamic transactions. The performance of RTC is compared to EQF. It is observed that EQF performs better than RTC especially when the number of processors is increased. The effect of ME (Mean of task execution time), MF (Mean of transaction flexibility), CVE (Coefficient of variation of task execution), and CVF (Coefficient of variation of transaction flexibility) on the relative performance of the algorithms is investigated. In general it is concluded that WP covers the greatest number of solutions since it largely subsumes EQF which in turn subsumes RTC and LM. The greatest coverage of the solution space might be achieved by using WP in conjunction with HOPA and EQF.

In Chapter 5, EQF, WP and RTC are extended to solve more complex real-time models (i.e., when transactions with subtransactions or revisiting transactions exist in the system). For revisiting transactions only WP and EQF algorithm are extended. In the case of revisiting transactions, as opposed to the flow shop, the worst case response times of the tasks are not identical at the current worst-case position (the current position with highest response time); WP is modified to overcome this problem. Both EQF and WP are modified to use a different algorithm for computing task worst-case response time (defined in

Chapter 3 for revisiting). The algorithms are also extended to solve transactions with subtransactions. The algorithms are extended in order to (1) map the complex transaction end-to-end deadlines to the subtransaction's end-to-end deadlines and, (2) handle tasks that are shared by different subtransactions. The relative performance of the extended algorithms is compared. In the case of transactions with subtransactions, the effect of number of processors in the parallel stage on the relative performance of algorithms is investigated. It is concluded that EWP performs better than EEQF and ERTC, especially at low values of CVE. For the revisiting case the effect of revisiting on the relative performance of EEQF and EWP is studied and it is concluded that EWP performs better than EEQF when revisiting transactions exist in the system. The effect of the revisiting on the relative performance of algorithms is investigated through the number of tasks in transactions. For a given number of processors, by increasing number of tasks in transactions, the number of revisiting tasks in a problem instance is increased. In this chapter the extended algorithms are also used to assign priorities to different tasks of a signal processor module in a towed array sonar system; the algorithms are applied to two different scenarios (i.e., problem instances). Solutions were found by EWP and EEQF for both scenarios; ERTC was only able to find solution for one of the scenarios.

6.2 Summary of Contributions

The main contributions of this thesis are listed below.

- An algorithm for computing a bound on the end-to-end response time of revisiting transactions is introduced.
- A problem with the LM algorithm is identified (i.e., LM algorithm gives all of the slack time to the tasks on the processor being scheduled and it does not consider that portion of the slack time that belongs to future activities of a transaction) and a new heuristic

algorithm (EQF) that alleviates the problem is introduced. Through simulation, the relative performance of EQF and LM is compared and it is shown that EQF is able to find feasible solutions for many problem instances in which LM is not successful.

- A problem with EQF is identified (i.e., a task may receive a priority such that it consumes a substantial slack time and insufficient slack time remains for future activities). To overcome this problem a new heuristic algorithm (WP) is introduced. Through simulation, it is shown that for many problem instances in which EQF fails WP is successful in finding feasible solutions (especially when the load is distributed on the smaller number of processors and processors are highly utilized).
- The performance of WP is compared to an existing heuristic (HOPA) and it is observed that for many problem instances in which HOPA is not successful in finding a feasible solution, WP can find a solution. It is also shown that by increasing number of processors HOPA is less able to solve problem instances.
- In [17] an algorithm for establishing real-time channels is proposed. The algorithm is modified in this research (i.e., a transaction's remaining slack time is divided among its tasks in proportion to the tasks minimum response times rather than divided equally among them) and is used for assigning priority to tasks. It is shown that modified algorithm (RTC: real-time channel) has a better performance compared to the one proposed in [17].
- The performance of RTC algorithm is compared to EQF. It is shown that EQF performs better than RTC especially as the number of processors is increased.
- EQF, WP, and RTC are extended to solve complex transactions (i.e., revisiting transactions or transactions with subtransactions). It is shown that extended WP algorithm has the best overall performance.

- Through simulation the effect of the workload parameters on the relative performance of the algorithms is studied and substantial insight into the behavior of the algorithms is gained.

6.3 Application Considerations

This thesis has been motivated largely by multiprocessor real-time systems that can be modeled as a set of periodic transactions with end-to-end deadlines. Typical of these applications are signal processing in sonar or radar systems or distributed control systems. Although end-to-end deadline timing requirements may be adequate for some of these systems, other systems may have different or additional timing requirements, for example jitter [39]. In this thesis, priorities are assigned to tasks only to satisfy the end-to-end deadline of transactions. Further modifications of the scheduling approach are required for handling jitter. However this is beyond the scope of this thesis.

In some real-time systems, specific tasks may need to run at predefined priority levels: for example a timer task may have to be run at the highest priority. In these cases, the effect of these tasks must be considered during task's priority assignment.

Since task execution times are difficult to predict, assumption of task execution times must be verified by measurement on the finished system to be sure that assumptions are not violated. This may be challenging, thus proper instrumentation should be considered during the design.

6.4 Future Work

Many issues addressed in this thesis remain open or can be extended. Some possible extensions to this work are listed below.

Improving Heuristics

In this research several new heuristics are developed and their drawbacks are identified. In EQF the slack time of a transaction's task is determined by the task's execution time, the remaining work in the transaction and the transaction's remaining slack time. The current remaining work of a transaction is computed by summing up the execution times of the transactions unscheduled tasks. However, summing task execution times does not fully reflect the remaining work of a transaction and better measures may be explored. The following example provides motivation for the investigation.

Figure 6.1 shows a real-time system consisting of three transactions and two processors. In this example three tasks are allocated to the first processor and one task to the second processor.

Figure 6.1 A Problem with EQF and WP

Applying EQF to this example, the local deadline for task *T1* is $16 \cdot (2/8) = 4$. At the time of assigning a local deadline to *T1*, EQF considers the summation of execution times for *T1* and *T4* ($2+6=8$) as the remaining work of the transaction and consequently part of the transaction slack time is assigned to *T4*. The algorithm does not take into account that task *T4* will not use its slack time because it is the only task allocated to processor *P2*. The performance of EQF can be improved by defining a better measure for the remaining work of a transaction. The WP algorithm also suffers from the same problem. For computing the

flexibility of a transaction, the remaining work of a transaction must be determined and defining a better measure will improve the performance of the algorithm.

RTC can be used for establishing dynamic transactions. It is shown in Chapter 4 that the division of the remaining slack time among the transaction's tasks has a substantial effect on the performance of RTC. In Chapter 4 an algorithm for dividing remaining slack time is introduced. More elegant algorithms might be developed to improve the performance of RTC. In order to apply RTC to the revisiting transactions, the problem of determining the minimum response time of these transactions must be solved. This problem is defined in Chapter 5 and remains unsolved.

Allocation Problem

Another issue that needs to be addressed is the problem of allocating tasks to processors. A good allocation policy increases schedulability (i.e., it will increase the number of feasible solutions) and consequently priority assignment algorithms will have a better chance to find a feasible solution. Research needs to be done to develop such efficient allocation algorithms to maximize the number of feasible solutions.

A Different look at the Timing Constraints of Transactions

In this thesis, it is assumed that each instance of a transaction has an end-to-end deadline by which it is expected to complete. This assumption is reasonable for many real-time systems. However there are real-time applications in which it is not necessary for every instance of a periodic transaction to meet its deadline. As an example, consider the playback of digitized video. A source (e.g., a video file server) generates a stream of video frames at a rate of 30 frames per second. Each frame has a deadline before which it must reach the destination. A frame that misses its deadline is considered lost. In this application, one can tolerate a few missed deadlines without significant degradation in the video quality. The timing requirements of such a stream can be expressed by specifying two con-

stants n and m such that in any window of n consecutive frames at least m of them meet their deadlines. This new definition for timing constraints of transactions can also help in graceful degradation of performance in overloaded situations. To make this concept applicable two problems must be solved. These problems are (1) computing the worst-case response time of a task and (2) mapping a transaction's timing constraint to its constituent tasks.

Combining Revisiting and Subtransactions

This research did not explore the effects of problems in which transactions with subtransactions also included revisiting and vice-versa. It would be worthwhile trying to understand whether such problems can be adequately solved by the existing algorithms, which algorithms had the best coverage, and why. A significant challenge in this task, would be the construction of the experimental model.

References

- [1] N. Audsley et al, "Applying new Scheduling Theory to Static Priority Preemptive Scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284-292, September 1993.
- [2] R. Bettati and J. W. S. Liu, "End-To-End Scheduling to Meet Deadlines in Distributed Systems," in *12th International Conference on Distributed Computer Systems*, pp. 452-492, IEEE Computer Society press, May 1992.
- [3] R. Bettati, *End-to-End Scheduling to Meet Deadlines in Distributed Systems*, Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, March 1994.
- [4] M. Chen and K. Lin, "Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems," *Journal of Real-time Systems*, vol. 1, no. 2, pp. 324-346, 1990.
- [5] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 466-473, 1989.
- [6] S. Cheng, J. A. Stankovic and K. Ramamritham, "Dynamic Scheduling of Groups of Tasks with Precedence Constraints in Distributed Hard Real-Time Systems," in *7th Real-Time System Symposium*, pp. 166-174, IEEE Computer Society Press, December 1986.
- [7] R. I. Davis et al, "Scheduling Slack Time in Fixed Priority Preemptive Systems," in *14th Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 222-231, December 1993.

- [8] M. Dertouzos, "Control Robotics: The Procedural Control of Physical Processes," in *Proceeding of the IFIP Congress*, pp. 807-813, August 1974.
- [9] S. K. Dhall and C. L. Liu, "On a Real-time Scheduling Problem," *Operations Research*, vol. 26, no. 1, pp. 127-140, 1978.
- [10] G. Fohler, "Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems," in *16th Real-Time System Symposium*, IEEE Computer Society Press, December 1995.
- [11] G. Fohler, "Scheduling of Pipelined Periodic Tasks in Distributed Real-Time Systems," CS-95-091, Department of Computer Science, University of Massachusetts at Amherst, Amherst, USA, October 1995.
- [12] J. J. G. Garcia, M. G. Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems," in *3th Workshop on Parallel and Distributed Real-time Systems*, IEEE Computer Society Press, pp. 124-131, April 1995.
- [13] J. Harista, M. Carey, and M. Livny, "Earliest Deadline Scheduling for Real-Time Database Systems," in *12th Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 232-242, December 1991.
- [14] K. S. Hong, J. Y-T. Leung, "On-Line Scheduling of Real-Time Tasks," in *9th Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 244-250, December 1988.
- [15] K. Jeffay, "The Real-time Producer/Consumer Paradigm: A Paradigm for the Construction of Efficient, Predictable Real-time Systems," in *ACM/SIGAPP Symposium on Applied Computing*, pp. 796-804, February 1993.

- [16] K. Jeffay, "Scheduling Sporadic Tasks with Shared Resources in Hard Real-time Systems," in *13th Real-time System Symposium*, IEEE Computer Society Press, pp. 89-99, December 1992.
- [17] D. D. Kandlur, K. G. Shin, D. Ferrari, "Real-time Communication in Multi-hop Networks," in *11th International Conference on Distributed Computer Systems*, pp. 300-307, IEEE Computer Society Press, May 1991.
- [18] H. Kopetz, A. Demm, C. Koza, and M. Mulozzani, "Distributed Fault Tolerant Real-time Systems: The Mars Approach," *IEEE Micro*, vol. 9, no. 1, pp. 25-40, February 1989.
- [19] J. Leung and J. Whitehead, "On the Complexity of Fixed-priority Scheduling of Periodic Real-Time Tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237-250, December 1982.
- [20] J. Leung and M. L. Merril, "A Note on Preemptive Scheduling of Periodic Real-Time Tasks," *Information Processing Letters*, vol. 11, no. 3, November 1980.
- [21] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 40-61, 1973.
- [22] A. K. Mok, *Fundamental Design Problems of Distributed Systems for Hard Real-time Environments*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, May 1983.

- [23] R. Nagarajan, *Quality-of-service Issues in High Speed Networks*. Ph.D. thesis, University of Massachusetts at Amherst, Amherst, USA, September 1993.
- [24] D. Niehaus, "Program Representation and Translation for Predictable Real-time Systems," in *12th Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 53-63, December 1989.
- [25] S. Pingali, *Protocol and Real-Time Scheduling Issues for Multimedia Applications*, Ph.D. thesis, Department of Electrical and Computer Engineering, University of Massachusetts at Amherst, Amherst, Massachusetts, USA, September 1993.
- [26] K. Ramamritham, "Allocation and Scheduling of Complex Periodic Tasks," in *10th International Conference on Distributed Computing Systems*, IEEE Computer Society, pp. 108-115, June 1990.
- [27] K. Ramamritham and J. M. Adan, "Providing for Dynamic Arrivals during Static Allocation and Scheduling of Periodic Tasks," CS-90-107, Department of Computer Science, University of Massachusetts at Amherst, Amherst, USA, October 1990.
- [28] R. Rajkumar, L. Sha, and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-time Synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175-1185, September 1990.
- [29] R. Rajkumar, L. Sha, and J. Lehoczky, "Real-Time Synchronization Protocols for Multiprocessors," in *9th Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 259-269, December 1988.

- [30] S. Ramos-Thuel and J. Lehoczky, "On-line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems," in *14th Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 160-171, December 1993.
- [31] N. Sadeh, K. Katia, and Yalin Xiong, "Backtracking Techniques for the Job Shop Constraint Satisfaction Problem," *Artificial Intelligence*, vol. 76, pp. 455-480, 1995.
- [32] K. S. Schwan and H. Zhou, "Dynamic Scheduling of Hard Real-time Tasks and Real-Time Threads," *IEEE Transactions on Software Engineering*, vol. 18, no. 8, pp. 736-747, August 1992.
- [33] J. A. Stankovic, "Two Extended Editorials on Real-time Kernels and Resource Allocation," CS-92-078, Department of Computer Science, University of Massachusetts at Amherst, Amherst, Massachusetts, USA, 1992
- [34] J. A. Stankovic, "Misconception about Real-Time Computing," *IEEE Computer*, vol. 21, no. 10, pp. 10-19, October 1988.
- [35] J. A. Stankovic, "The Spring Kernel: A New Paradigm for Hard Real-time Operating System," *IEEE Software*, vol. 8, no. 3, pp. 62-72, May 1991.
- [36] J. A. Stankovic, K. Ramamritham, and S. Cheng, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems," *IEEE Transactions on Computers*, vol. 34, no. 12, pp. 1130-1142, December 1985.
- [37] T. S. Tia et al, "A Linear-Time Optimal Acceptance Test for Scheduling of Hard Real-Time Tasks," *Submitted to IEEE Transactions on Software Engineering*.

- [38] J. Wiest and F. Levy, *A Management Guide to Pert/CPM*. 2d edition, Prentice-Hall publication, 1977.
- [39] B. Wittenmark, J. Nilsson, and M. Torngren, "Timing Problems in Real-Time Control Systems," in *American Control Conference*, IEEE Press, pp. 2000-2004, June 1995.
- [40] J. Xu and D. L. Parnas, "On Satisfying Timing Constraints in Hard Real-Time Systems," *IEEE Transactions on Software Engineering*, vol. 19, no. 1, pp. 70-84, January 1993.
- [41] J. Xu and D. L. Parnas, "Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations," *IEEE Transactions on Software Engineering*, vol. 16, no. 3, pp. 360-369, March 1990.
- [42] J. Xu, "Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence and Exclusion Relations," *IEEE Transactions on Software Engineering*, vol. 19, no. 2, pp. 139-154, February 1993.
- [43] W. Zhao, K. Ramamritham, and J. A. Stankovic, "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems," *IEEE Transactions on Software Engineering*, vol. 13, no. 5, pp. 564-577, May 1987.
- [44] Q. Zheng and K. G. Shin, "On the Ability of Establishing Real-time Channels in Point-To-Point Packet-Switched Networks," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 1096-1105, 1994.

Appendix A

Simulation Results for Algorithm Comparison

In this appendix, the raw simulation results of the experiments defined in Table 4.6 are presented. The experiments are designed to study the effect of CVE (Coefficient of variation of task execution) time, CVF (Coefficient of variation of transaction flexibility), ME (Mean of task execution time), MF (Mean of transaction flexibility), and NT (Number of transactions) on the relative performance of following pair of algorithms: [WP,EQF], [WP,HOPA],[EQF,RTC], and [EQF,LM]. Note that in all the tables, the column **UTIL>1.0** reports the number of problem instances which do not have a feasible solution (i.e., the utilization in at least one of the processors is greater or equal to 1.0).

CVE	Only EQF	Only WP	EQF & WP	Solutions Found	Util >1.0
0.0	0	1	0	50000	0
0.25	0	4e-05	0.99996	50000	0
0.50	0	0.000360144	0.99964	49980	0
0.75	2.02037e-05	0.00468725	0.995293	49496	4
1.00	8.24334e-05	0.00869673	0.991221	48524	84
1.25	0.000105722	0.00981097	0.990083	47294	421
1.50	0.000129853	0.0101069	0.989763	46206	970
1.75	0.000154015	0.00807481	0.991771	45450	1681
2.00	0.000111889	0.00608678	0.993801	44687	2438

Table A.1 CVF=0.0 MF=2.5 NT=5

CVE	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	0.0325633	0.123321	0.844116	43853	0
0.25	0.0139251	0.0401821	0.945893	48330	0
0.50	0.00486677	0.0121467	0.982987	49314	1
0.75	0.0020725	0.0157924	0.982135	48251	62
1.00	0.00124189	0.01835	0.980408	46703	403
1.25	0.000972225	0.0187816	0.980246	45257	1216
1.50	0.000660788	0.0141728	0.985166	43887	2121
1.75	0.000463994	0.0116463	0.98789	43104	3108
2.00	0.000401076	0.00858774	0.991011	42386	3939

Table A.2 CVF=0.25 MF=2.5 NT=5

CVE	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	0.00774003	0.174476	0.817784	33850	0
0.25	0.025983	0.0691992	0.904818	37486	30
0.50	0.0247387	0.0438029	0.931458	40180	357
0.75	0.0138931	0.0498295	0.936277	39876	1508
1.00	0.00870808	0.0460619	0.94523	38470	3229
1.25	0.00494421	0.0407154	0.95434	37013	5284
1.50	0.00447177	0.0295416	0.965987	35780	7123
1.75	0.00266774	0.0231491	0.974183	34861	8745
2.00	0.00177558	0.017552	0.980672	34355	10024

Table A.3 CVF=0.5 MF=2.5 NT=5

CVE	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	0	1	0	49871	0
0.25	2e-05	0.02304	0.97694	50000	0
0.50	0	0.00178475	0.998215	49867	0
0.75	8.15627e-05	0.0107051	0.989213	49042	0
1.00	6.25091e-05	0.0165858	0.983352	47993	50
1.25	0.000128066	0.0179719	0.9819	46851	269
1.50	6.5341e-05	0.0162699	0.983665	45913	755
1.75	0.000176511	0.0133928	0.986431	45323	1225
2.00	0.000200401	0.010265	0.989535	44910	1904

Table A.4 CVF=0.0 MF=5.0 NT=10

CVE	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	0.0175624	0.359309	0.623128	33993	0
0.25	0.00657107	0.157749	0.83568	45959	0
0.50	0.0033465	0.035793	0.960861	48110	1
0.75	0.00111349	0.0384582	0.960428	46700	68
1.00	0.000755253	0.038007	0.961238	45018	420
1.25	0.000731629	0.0326718	0.966597	43738	1196
1.50	0.000799737	0.028179	0.971021	42514	2207
1.75	0.000621801	0.0207586	0.97862	41814	3171
2.00	0.000291517	0.0171509	0.982558	41164	4130

Table A.5 CVF=0.0 MF=5.0 NT=10

CVE	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	0.000718391	0.482175	0.517107	22272	88
0.25	0.00608686	0.218567	0.775346	26779	438
0.50	0.0167352	0.104037	0.879228	30893	1781
0.75	0.0116821	0.102107	0.886211	31330	4201
1.00	0.00809902	0.0948695	0.897031	30621	6920
1.25	0.00551365	0.0782735	0.916213	29563	9643
1.50	0.00389891	0.0602938	0.935807	28726	12078
1.75	0.00362074	0.0454723	0.950907	28171	13909
2.00	0.00248891	0.0341233	0.963388	27723	15269

Table A.6 CVF=0.0 MF=5.0 NT=10

NT	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0	0	1	50000	0
3	0	0	1	50000	0
4	0	0	1	50000	0
5	0	0.00046011	0.99954	49988	0
6	0.0119405	0.0663082	0.921751	47988	1

Table A.7 CVE=0.5 CVF=0.0 MF=2.5

NT	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0	0	1	50000	0
3	0	0	1	49996	0
4	0	0.00183936	0.998161	48930	362
5	4.47057e-05	0.00603527	0.99392	44737	2351
6	0.0012913	0.0192291	0.97948	35623	7656

Table A.8 CVE=2.0 CVF=0.0 MF=2.5

NT	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0	0	1	49957	0
3	0.000687619	0.00323585	0.996077	49446	0
4	0.00245231	0.0399965	0.957551	46079	0
5	0.00774003	0.174476	0.817784	33850	0
6	0.00836361	0.410434	0.581202	14587	26

Table A.9 CVE=0.0 CVF=0.5 MF=2.5

NT	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
7	0	0	1	50000	0
8	0	0	1	50000	0
9	0	6e-05	0.99994	50000	0
10	0	0.00150364	0.998496	49879	0
11	0.000640549	0.0357674	0.963592	48396	0
12	0.0228277	0.364992	0.61218	35746	2

Table A.10 CVE=0.5 CVF=0.0 MF=5.0

NT	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
7	0	0.000340784	0.999659	49885	11
8	0	0.00190461	0.998095	49354	148
9	2.08524e-05	0.00506714	0.994912	47956	577
10	0.000200709	0.0110167	0.988783	44841	1866
11	0.000226632	0.0216056	0.978168	39712	4341
12	0.00126175	0.0404391	0.958299	31702	8694

Table A.11 CVE=2.0 CVF=0.0 MF=5.0

NT	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
7	0.00051623	0.0441687	0.955315	48428	0
8	0.000382478	0.133192	0.866425	44447	2
9	0.000957019	0.292594	0.706449	35527	6
10	0.000718391	0.482175	0.517107	22272	88
11	0.00140351	0.654135	0.344461	9975	422
12	0.000657678	0.786912	0.21243	3041	1632

Table A.12 CVE=0.0 CVF=0.5 MF=5.0

MF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
1.5	0.228571	0.43354	0.337888	805	450
1.7	0.192243	0.336249	0.471508	15548	54
1.9	0.0756929	0.165197	0.75911	42646	3
2.1	0.00511821	0.0285157	0.966366	49236	0
2.3	0.000100227	0.00292661	0.996973	49887	0
2.5	0	0.00046011	0.99954	49988	0
2.7	0	2e-05	0.99998	50000	0
2.9	0	0	1	50000	0
3.1	0	0	1	50000	0
3.3	0	0	1	50000	0
3.5	0	0	1	50000	0

Table A.13 CVE=0.5 CVF=0.0 NT=5

MF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
1.5	0.0074329	0.0438403	0.948727	14530	23188
1.7	0.00112602	0.03318	0.962694	23267	16157
1.9	0.00189133	0.0225036	0.975605	31195	10519
2.1	0.000799254	0.01428	0.984921	37535	6612
2.3	0.000477555	0.00993314	0.989589	41880	4091
2.5	4.47057e-05	0.00603527	0.99392	44737	2351
2.7	0.000106934	0.00470508	0.995188	46758	1229
2.9	0	0.00328366	0.996716	48117	586
3.1	0	0.00177732	0.998223	48950	261
3.3	0	0.00101071	0.998989	49470	113
3.5	0	0.000542866	0.999457	49736	43

Table A.14 CVE=2.0 CVF=0.0 NT=5

MF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
1.5	0.028314	0.528958	0.442728	1554	161
1.7	0.0205177	0.457143	0.522339	5215	39
1.9	0.018264	0.369803	0.611933	11498	13
2.1	0.0137368	0.296065	0.690198	19364	1
2.3	0.00845518	0.230062	0.761483	27084	0
2.5	0.00774003	0.174476	0.817784	33850	0
2.7	0.00481372	0.128511	0.866675	39055	0
2.9	0.00357301	0.0939959	0.902431	42821	0
3.1	0.0024472	0.0647956	0.932757	45358	0
3.3	0.00176277	0.0450674	0.95317	47085	0
3.5	0.00114146	0.0307571	0.968101	48184	0

Table A.15 CVE=0.0 CVF=0.5 NT=5

MF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
4.0	0.0566465	0.427564	0.515789	27804	8
4.2	0.0175729	0.224191	0.758237	42281	1
4.4	0.00295359	0.0745781	0.922468	47400	0
4.6	0.000285371	0.0203429	0.979372	49059	0
4.8	2.0143e-05	0.0053379	0.994642	49645	0
5.0	0	0.00150364	0.998496	49879	0
5.2	0	0.000420261	0.99958	49969	0
5.4	0	8.00176e-05	0.99992	49989	0
5.6	0	2e-05	0.99998	50000	0
5.8	0	0	1	50000	0
6.0	0	0	1	50000	0

Table A.16 CVE=0.5 CVF=0.0 NT=10

MF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
4.0	0.00218679	0.0419723	0.955841	28352	11451
4.2	0.001447	0.0308694	0.967684	33172	8273
4.4	0.000591589	0.0245778	0.974831	37188	5925
4.6	0.000371407	0.0185456	0.981083	40387	4114
4.8	0.000302804	0.0141619	0.985535	42932	2813
5.0	0.000200709	0.0110167	0.988783	44841	1866
5.2	0.000108012	0.00859778	0.991294	46291	1187
5.4	0	0.00641716	0.993583	47373	714
5.6	0	0.00452188	0.995478	48210	431
5.8	2.04935e-05	0.00360685	0.996373	48796	264
6.0	0	0.00241919	0.997581	49190	158

Table A.17 CVE=2.0 CVF=0.0 NT=10

MF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
4.0	0.00248988	0.737317	0.260193	3213	1284
4.2	0.00177274	0.687467	0.310761	5641	717
4.4	0.00133645	0.641942	0.356721	8979	405
4.6	0.00160342	0.592808	0.405589	13097	231
4.8	0.00124773	0.53936	0.459392	17632	138
5.0	0.000718391	0.482175	0.517107	22272	88
5.2	0.000667705	0.430262	0.56907	26958	46
5.4	0.000607922	0.375088	0.624304	31254	23
5.6	0.000456582	0.325029	0.674514	35043	11
5.8	0.00049523	0.277616	0.721889	38366	7
6.0	0.000413897	0.23132	0.768266	41073	5

Table A.18 CVE=0.0 CVF=0.5 NT=10

CVF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	0	0.00046011	0.99954	49988	0
0.1	4.0032e-05	0.000860689	0.999099	49960	0
0.2	0.00215305	0.00682134	0.991026	49697	0
0.3	0.0111697	0.0215089	0.967321	48166	4
0.4	0.0198107	0.0332351	0.946954	44471	63
0.5	0.0252986	0.0444612	0.93024	39765	384

Table A.19 CVE=0.5 MF=2.5 NT=5

CVF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	8.23995e-05	0.00861074	0.991307	48544	78
0.1	0.000289903	0.0101673	0.989543	48292	127
0.2	0.000822941	0.0163322	0.982845	47391	268
0.3	0.00228909	0.0257522	0.971959	45433	679
0.4	0.00536339	0.0394339	0.955203	42324	1601
0.5	0.00904954	0.0473259	0.943625	38013	3425

Table A.20 CVE=1.0 MF=2.5 NT=5

CVF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	4.47057e-05	0.00603527	0.99392	44737	2351
0.1	0.000179909	0.00746621	0.992354	44467	2575
0.2	0.000231599	0.00914818	0.99062	43178	3487
0.3	0.00073434	0.0117005	0.987565	40853	5054
0.4	0.00126743	0.0137305	0.985002	37872	7208
0.5	0.00179507	0.0180978	0.980107	33982	10304

Table A.21 CVE=2.0 MF=2.5 NT=5

CVF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	0	0.00150364	0.998496	49879	0
0.1	2.00831e-05	0.00387605	0.996104	49793	0
0.2	0.00160906	0.0209789	0.977412	49097	0
0.3	0.00783892	0.0560142	0.936147	45542	18
0.4	0.014238	0.0861062	0.899656	38348	321
0.5	0.0173591	0.105121	0.87752	30013	1975

Table A.22 CVE=0.5 MF=5.0 NT=10

CVF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	8.31411e-05	0.0164827	0.983434	48111	49
0.1	8.40407e-05	0.0205269	0.979389	47596	78
0.2	0.000476005	0.0305941	0.96893	46218	248
0.3	0.00199735	0.0532782	0.944724	43057	867
0.4	0.00522746	0.0742836	0.920489	37303	2919
0.5	0.00832108	0.0994519	0.892227	29924	7331

Table A.23 CVE=1.0 MF=5.0 NT=10

CVF	Only EQF	Only WP	EQF & WP	Solutions Found	Util > 1.0
0.0	0.000200709	0.0110167	0.988783	44841	1866
0.1	0.000158231	0.0110988	0.988743	44239	2108
0.2	0.000259795	0.0145013	0.985239	42341	3257
0.3	0.000925164	0.0209447	0.97813	38912	5667
0.4	0.00166493	0.0286012	0.969734	33635	9835
0.5	0.00291707	0.0348571	0.962226	27082	15867

Table A.24 CVE=2.0 MF=5.0 NT=10

CVE	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	0	0	1	50000	0
0.25	0	0	1	50000	0
0.50	0	0.000320128	0.99968	49980	0
0.75	0	0.00327306	0.996727	49495	4
1.00	8.24334e-05	0.00525513	0.994662	48524	84
1.25	0.000105722	0.00414429	0.99575	47294	42
1.50	2.16446e-05	0.00411247	0.995866	46201	970
1.75	0.000132016	0.00312438	0.996744	45449	1681
2.00	0.000156638	0.00187966	0.997964	44689	2438

Table A.25 CVF=0.0 MF=2.5 NT=5

CVE	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	0.0469291	0.0288873	0.924181	44514	0
0.25	0.0227413	0.0103351	0.966924	48766	0
0.50	0.00462456	0.0103444	0.985031	49302	1
0.75	0.00147236	0.0138319	0.984696	48222	62
1.00	0.00107078	0.0123782	0.986551	46695	405
1.25	0.00095015	0.0101865	0.988863	45256	1216
1.50	0.000478589	0.00683698	0.992684	43879	2121
1.75	0.000440805	0.00433845	0.995221	43103	3108
2.00	0.000424659	0.00304339	0.996532	42387	3939

Table A.26 CVF=0.25 MF=2.5 NT=5

3

OF/DE

3

PM-1 3 1/2" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

CVE	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	0.0209007	0.0341641	0.944935	34305	0
0.25	0.0451134	0.0295525	0.925334	38237	30
0.50	0.0260718	0.0367839	0.937144	40235	357
0.75	0.013423	0.0409213	0.945656	39857	1508
1.00	0.00883691	0.0328005	0.958363	38475	3229
1.25	0.00588426	0.0250216	0.969094	37048	5284
1.50	0.00474993	0.0160101	0.97924	35790	7123
1.75	0.00329673	0.0117249	0.984978	34883	8745
2.00	0.00232734	0.00770932	0.989963	34374	10024

Table A.27 CVF=0.5 MF=2.5 NT=3

CVE	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	0.00160157	0.0568557	0.941543	49951	0
0.25	2e-05	0	0.99998	50000	0
0.50	0	0.00134357	0.998656	49867	0
0.75	6.11733e-05	0.00552599	0.994413	49041	0
1.00	0.000104178	0.00654235	0.993353	47995	50
1.25	0.000256098	0.00608234	0.993662	46857	269
1.50	0.000195997	0.00407239	0.995732	45919	755
1.75	0.000308853	0.00253701	0.997154	45329	1225
2.00	0.000267183	0.0020484	0.997684	44913	1904

Table A.28 CVF=0.0 MF=5.0 NT=10

CVE	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	0.00922657	0.219806	0.770967	33707	0
0.25	0.0117746	0.0454319	0.942793	46201	0
0.50	0.0026624	0.027248	0.97009	48077	1
0.75	0.00107071	0.0223992	0.97653	46698	68
1.00	0.000999356	0.018055	0.980946	45029	420
1.25	0.000937207	0.0134866	0.985576	43747	1196
1.50	0.00115216	0.00919373	0.989654	42529	2207
1.75	0.000932412	0.00595309	0.993114	41827	3171
2.00	0.000607135	0.00356995	0.995823	41177	4130

Table A.29 CVF=0.0 MF=5.0 NT=10

CVE	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	0.000718391	0.208109	0.791173	22272	88
0.25	0.0101897	0.129491	0.86032	26890	438
0.50	0.0131895	0.0931388	0.893672	30782	1781
0.75	0.0119974	0.0708679	0.917135	31340	4201
1.00	0.0102001	0.0519781	0.937822	30686	6920
1.25	0.00809717	0.0367409	0.955162	29640	9643
1.50	0.00694107	0.023322	0.969737	28814	12078
1.75	0.00517455	0.0161616	0.978664	28215	13909
2.00	0.00406958	0.0105161	0.985414	27767	15269

Table A.30 CVF=0.0 MF=5.0 NT=10

NT	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0	0	1	50000	0
3	0	0	1	50000	0
4	0	0	1	50000	0
5	0	0.000280067	0.99972	49988	0
6	0.010621	0.0477005	0.941678	47924	1

Table A.31 CVE=0.5 CVF=0.0 MF=2.5

NT	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0	0	1	50000	0
3	0	0	1	49996	0
4	0	0.000367872	0.999632	48930	362
5	6.70571e-05	0.00183289	0.9981	44738	2351
6	0.00168364	0.0075764	0.99074	35637	7656

Table A.32 CVE=2.0 CVF=0.0 MF=2.5

NT	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	4.00328e-05	2.00164e-05	0.99994	49959	0
3	0.00099068	0.00131417	0.997695	49461	0
4	0.0058181	0.00932194	0.98486	46235	0
5	0.0209007	0.0341641	0.944935	34305	0
6	0.0324415	0.101739	0.865819	14950	26

Table A.33 CVE=0.0 CVF=0.5 MF=2.5

NT	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
7	0	0	1	50000	0
8	0	0	1	50000	0
9	0	6e-05	0.99994	50000	0
10	0	0.00106257	0.998937	49879	0
11	0.000248052	0.0252186	0.974533	48377	0
12	0.014613	0.283796	0.701591	35448	2

Table A.34 CVE=0.5 CVF=0.0 MF=5.0

NT	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
7	0	6.01383e-05	0.99994	49885	11
8	2.02614e-05	0.000162091	0.999818	49355	148
9	4.1704e-05	0.000813229	0.999145	47957	577
10	0.000312179	0.00209606	0.997592	44846	1866
11	0.000553807	0.00511013	0.994336	39725	4341
12	0.00277165	0.0120945	0.985134	31750	8694

Table A.35 CVE=2.0 CVF=0.0 MF=5.0

NT	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
7	0.000970072	0.0099484	0.989082	48450	0
8	0.00157303	0.0364494	0.961978	44500	2
9	0.00160338	0.104895	0.893502	35550	6
10	0.000718391	0.208109	0.791173	22272	88
11	0.00100291	0.3448	0.654197	9971	422
12	0.000657678	0.491943	0.507399	3041	1632

Table A.36 CVE=0.0 CVF=0.5 MF=5.0

MF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
1.5	0.285386	0.315305	0.39931	869	450
1.7	0.213489	0.281438	0.505073	15968	54
1.9	0.0749554	0.135525	0.789519	42612	3
2.1	0.00410686	0.0255561	0.970337	49186	0
2.3	0	0.00332785	0.996672	49882	0
2.5	0	0.000280067	0.99972	49988	0
2.7	0	2e-05	0.99998	50000	0
2.9	0	0	1	50000	0
3.1	0	0	1	50000	0
3.3	0	0	1	50000	0
3.5	0	0	1	50000	0

Table A.37 CVE=0.5 CVF=0.0 NT=5

MF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
1.5	0.00981806	0.0257467	0.964435	14565	23188
1.7	0.00489586	0.0180374	0.977067	23285	16157
1.9	0.00205128	0.0114423	0.986506	31200	10519
2.1	0.0011186	0.00671159	0.99217	37547	6612
2.3	0.000477555	0.00396371	0.995559	41880	4091
2.5	6.70571e-05	0.00183289	0.9981	44738	2351
2.7	6.41629e-05	0.00126187	0.998674	46756	1229
2.9	2.07822e-05	0.00083129	0.999148	48118	586
3.1	0	0.000245148	0.999755	48950	261
3.3	2.02139e-05	0.000141497	0.999838	49471	113
3.5	0	0.000120637	0.999879	49736	43

Table A.38 CVE=2.0 CVF=0.0 NT=5

MF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
1.5	0.0936375	0.12425	0.782113	1666	161
1.7	0.0714416	0.0985275	0.830031	5501	39
1.9	0.0553975	0.0776569	0.866946	11950	13
2.1	0.038659	0.0632236	0.898117	19866	1
2.3	0.0275565	0.0455171	0.926926	27616	0
2.5	0.0209007	0.0341641	0.944935	34305	0
2.7	0.0152026	0.0256417	0.959156	39467	0
2.9	0.0104594	0.0181822	0.971358	43119	0
3.1	0.00656479	0.0115488	0.981886	45546	0
3.3	0.00474315	0.00891458	0.986342	47226	0
3.5	0.00296238	0.00625621	0.990781	48272	0

Table A.39 CVE=0.0 CVF=0.5 NT=5

MF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
4.0	0.0456629	0.361119	0.593218	27484	8
4.2	0.013396	0.158686	0.827918	42102	1
4.4	0.00141568	0.0488305	0.949754	47327	0
4.6	0.000122322	0.0142709	0.985607	49051	0
4.8	0	0.00410926	0.995891	49644	0
5.0	0	0.00106257	0.998937	49879	0
5.2	0	0.000220136	0.99978	49969	0
5.4	0	2.00044e-05	0.99998	49989	0
5.6	0	2e-05	0.99998	50000	0
5.8	0	0	1	50000	0
6.0	0	0	1	50000	0

Table A.40 CVE=0.5 CVF=0.0 NT=10

MF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
4.0	0.00397845	0.0156674	0.980354	28403	11451
4.2	0.00222905	0.0103621	0.987409	33198	8273
4.4	0.00104825	0.00709582	0.991856	37205	5925
4.6	0.000693069	0.00559406	0.993713	40400	4114
4.8	0.000395938	0.00363331	0.995971	42936	2813
5.0	0.000312179	0.00209606	0.997592	44846	1866
5.2	0.000129612	0.00144733	0.998423	46292	1187
5.4	0.000105534	0.00109756	0.998797	47378	714
5.6	0	0.000705248	0.999295	48210	431
5.8	2.04935e-05	0.00040987	0.99957	48796	264
6.0	0	0.000243952	0.999756	49190	158

Table A.41 CVE=2.0 CVF=0.0 NT=10

MF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
4.0	0.0034204	0.402363	0.594216	3216	1284
4.2	0.00230333	0.355422	0.642275	5644	717
4.4	0.0025584	0.323026	0.674416	8990	405
4.6	0.0022129	0.283403	0.714384	13105	231
4.8	0.00113443	0.246398	0.752467	17630	138
5.0	0.000718391	0.208109	0.791173	22272	88
5.2	0.00103827	0.179064	0.819898	26968	46
5.4	0.00108734	0.150788	0.848124	31269	23
5.6	0.000542145	0.122839	0.876619	35046	11
5.8	0.000599427	0.103623	0.895778	38370	7
6.0	0.000535566	0.0825259	0.916939	41078	5

Table A.42 CVE=0.0 CVF=0.5 NT=10

CVF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	0	0.00046011	0.99954	49988	0
0.1	4.0032e-05	0.000860689	0.999099	49960	0
0.2	0.00215305	0.00682134	0.991026	49697	0
0.3	0.0111697	0.0215089	0.967321	48166	4
0.4	0.0198107	0.0332351	0.946954	44471	63
0.5	0.0252986	0.0444612	0.93024	39765	384

Table A.43 CVE=0.5 MF=2.5 NT=5

CVF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	0.000123594	0.00486137	0.995015	48546	78
0.1	0.000165679	0.00648221	0.993352	48286	127
0.2	0.000633152	0.0101093	0.989258	47382	268
0.3	0.00235496	0.0179373	0.979708	45436	679
0.4	0.00519886	0.027294	0.967507	42317	1601
0.5	0.00951855	0.0344719	0.95601	38031	3425

Table A.44 CVE=1.0 MF=2.5 NT=5

CVF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	6.70571e-05	0.00183289	0.9981	44738	2351
0.1	0.000179909	0.00229384	0.997526	44467	2575
0.2	0.000277906	0.00312645	0.996596	43180	3487
0.3	0.000881079	0.00413618	0.994983	40859	5054
0.4	0.00158378	0.00522648	0.99319	37884	7208
0.5	0.00226484	0.00773575	0.989999	33998	10304

Table A.45 CVE=2.0 MF=2.5 NT=5

CVF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	0	0.00106257	0.998937	49879	0
0.1	4.01655e-05	0.00283167	0.997128	49794	0
0.2	0.00116149	0.0156291	0.983209	49075	0
0.3	0.00537102	0.047965	0.946664	45429	18
0.4	0.0107037	0.0771767	0.91212	38211	321
0.5	0.0129192	0.0988018	0.888279	29878	1975

Table A.46 CVE=0.5 MF=5.0 NT=10

CVF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	8.31411e-05	0.00679678	0.99312	48111	49
0.1	4.20221e-05	0.00821532	0.991743	47594	78
0.2	0.000562503	0.0144736	0.984964	46222	248
0.3	0.00239123	0.0267911	0.970818	43074	867
0.4	0.00690467	0.0396885	0.953407	37366	2919
0.5	0.0102065	0.0544011	0.935392	29981	7331

Table A.47 CVE=1.0 MF=5.0 NT=10

CVF	Only HOPA	Only WP	HOPA& WP	Solutions Found	Util > 1.0
0.0	0.000312179	0.00209606	0.997592	44846	1866
0.1	0.000248627	0.00271229	0.997039	44243	2108
0.2	0.000590249	0.00339983	0.99601	42355	3257
0.3	0.00164355	0.00490498	0.993451	38940	5667
0.4	0.00267308	0.00775194	0.989575	33669	9835
0.5	0.00475453	0.0093985	0.985847	27132	15867

Table A.48 CVE=2.0 MF=5.0 NT=10

CVE	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	NaN	NaN	NaN	0	0
0.25	0.665193	2.00004e-05	0.334787	49999	0
0.50	0.0817041	0.000220119	0.918076	49973	0
0.75	0.0252849	0.00269247	0.972023	49397	4
1.00	0.0166836	0.00556118	0.977755	48371	84
1.25	0.0115152	0.00689216	0.981593	47155	421
1.50	0.00746269	0.00774471	0.984793	46096	970
1.75	0.0053562	0.00628196	0.988362	45368	1681
2.00	0.003383	0.00492887	0.991688	44635	2438

Table A.49 CVF=0.0 MF=2.5 NT=5

CVE	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	0.920022	7.80275e-05	0.0799001	38448	0
0.25	0.477898	0.000732411	0.521369	46422	0
0.50	0.176235	0.0014758	0.822289	48787	1
0.75	0.0818117	0.00559092	0.912597	47756	62
1.00	0.0429868	0.0091636	0.94785	46270	403
1.25	0.0236129	0.0117063	0.964681	44933	1216
1.50	0.013852	0.0094102	0.976738	43676	2121
1.75	0.010101	0.00847181	0.981427	42966	3108
2.00	0.00619429	0.00650164	0.987304	42297	3939

Table A.50 CVF=0.25 MF=2.5 NT=5

CVE	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	0.503183	0.000500751	0.496316	27958	0
0.25	0.321142	0.0014881	0.67737	34944	30
0.50	0.222225	0.00525594	0.772519	38623	357
0.75	0.137683	0.0142314	0.848085	38436	1508
1.00	0.0868205	0.0187438	0.894436	37399	3229
1.25	0.0521169	0.0219541	0.925929	36303	5284
1.50	0.0323758	0.0173199	0.950304	35335	7123
1.75	0.0202837	0.0160363	0.96368	34609	8745
2.00	0.0137784	0.0126375	0.973584	34184	10024

Table A.51 C VF=0.5 MF=2.5 NT=3

CVE	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	NaN	NaN	NaN	0	0
0.25	0.991607	2.04712e-05	0.00837274	48849	0
0.50	0.232298	0.000401623	0.7673	49798	0
0.75	0.0328528	0.00504481	0.962102	48763	0
1.00	0.0147216	0.0102338	0.975045	47685	50
1.25	0.00894543	0.0130213	0.978033	46616	269
1.50	0.00493978	0.0127866	0.982274	45751	755
1.75	0.00318563	0.0107736	0.986041	45203	1225
2.00	0.00238653	0.00860935	0.989004	44835	1904

Table A.52 C VF=0.0 MF=5.0 NT=10

CVE	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	0.993802	9.18231e-05	0.00610624	21781	0
0.25	0.794924	0.000593824	0.204482	38732	0
0.50	0.302154	0.00275174	0.695094	46516	1
0.75	0.107367	0.0122523	0.880381	45461	68
1.00	0.0454195	0.0189607	0.93562	44144	420
1.25	0.0227879	0.0191947	0.958017	43137	1196
1.50	0.011746	0.0196004	0.968654	42142	2207
1.75	0.00702192	0.0147738	0.978224	41560	3171
2.00	0.00404552	0.0140131	0.981941	41033	4130

Table A.53 CVF=0.0 MF=5.0 NT=10

CVE	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	0.610982	0.00586156	0.383157	11601	88
0.25	0.428687	0.00414029	0.567173	21013	438
0.50	0.32602	0.00606866	0.667912	27848	1781
0.75	0.193539	0.0207811	0.785679	28728	4201
1.00	0.104847	0.0355291	0.859623	28737	6920
1.25	0.0584457	0.0382932	0.903261	28334	9643
1.50	0.0353025	0.0364448	0.928253	28015	12078
1.75	0.0212306	0.0307465	0.948023	27743	13909
2.00	0.0139789	0.0252275	0.960794	27470	15269

Table A.54 CVF=0.0 MF=5.0 NT=10

NT	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0	0	1	50000	0
3	0	0	1	50000	0
4	0.0007	0	0.9993	50000	0
5	0.0888026	0.000220106	0.910977	49976	0
6	0.759768	0.00251564	0.237717	44919	1

Table A.55 CVE=0.5 CVF=0.0 MF=2.5

NT	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0	0	1	50000	0
3	0	0	1	49996	0
4	0.000592768	0.00169654	0.997711	48923	362
5	0.00351325	0.0049454	0.991541	44688	2351
6	0.015475	0.0133853	0.97114	35412	7656

Table A.56 CVE=2.0 CVF=0.0 MF=2.5

NT	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0.00566487	0	0.994335	49957	0
3	0.0600373	0	0.939963	49286	0
4	0.244456	0.000113017	0.75543	44241	0
5	0.503183	0.000500751	0.496316	27958	0
6	0.71412	0.00220443	0.283676	8619	26

Table A.57 CVE=0.0 CVF=0.5 MF=2.5

NT	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
7	0	0	1	50000	0
8	0.00058	0	0.99942	50000	0
9	0.0320019	0	0.967998	49997	0
10	0.259608	0.000501716	0.73989	49829	0
11	0.748546	0.00186088	0.249594	46752	0
12	0.977261	0.00162738	0.0211119	22736	2

Table A.58 CVE=0.5 CVF=0.0 MF=5.0

NT	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
7	0	0.000320744	0.999679	49884	11
8	0.000141853	0.00176303	0.998095	49347	148
9	0.000646763	0.0045482	0.994805	47931	577
10	0.00245717	0.00938191	0.988161	44767	1866
11	0.00536125	0.0174241	0.977215	39543	4341
12	0.0151007	0.0288287	0.956071	31323	8694

Table A.59 CVE=2.0 CVF=0.0 MF=5.0

NT	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
7	0.162914	0.000367123	0.836717	46306	0
8	0.311721	0.00119254	0.687087	38573	2
9	0.470954	0.0041211	0.524925	25236	6
10	0.610982	0.00586156	0.383157	11601	88
11	0.737644	0.00862069	0.253736	3480	422
12	0.821374	0.010687	0.167939	655	1632

Table A.60 CVE=0.0 CVF=0.5 MF=5.0

MF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
1.5	0.991228	0	0.00877193	456	450
1.7	0.96865	0.00145138	0.0298984	10335	54
1.9	0.872386	0.00324775	0.124367	35717	3
2.1	0.616232	0.00231525	0.381453	47943	0
2.3	0.289608	0.000944003	0.709448	49788	0
2.5	0.0888026	0.000220106	0.910977	49976	0
2.7	0.02624	2e-05	0.97374	50000	0
2.9	0.00722	0	0.99278	50000	0
3.1	0.00118	0	0.99882	50000	0
3.3	2e-05	0	0.99998	50000	0
3.5	0	0	1	50000	0

Table A.61 CVE=0.5 CVF=0.0 NT=5

MF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
1.5	0.0812659	0.0207922	0.897942	14188	23188
1.7	0.0479539	0.0175569	0.934489	22897	16157
1.9	0.0257555	0.0133631	0.960881	30906	10519
2.1	0.013942	0.00990126	0.976157	37369	6612
2.3	0.00734995	0.00730207	0.985348	41769	4091
2.5	0.00351325	0.0049454	0.991541	44688	2351
2.7	0.00194752	0.00402346	0.994029	46726	1229
2.9	0.00114369	0.00272406	0.996132	48090	586
3.1	0.000449466	0.00171614	0.997834	48947	261
3.3	0.00022237	0.000950128	0.998828	49467	113
3.5	0.000120639	0.000522771	0.999357	49735	43

Table A.62 CVE=2.0 CVF=0.0 NT=5

MF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
1.5	0.884038	0.00136426	0.114598	733	161
1.7	0.80677	0.00176305	0.191467	2836	39
1.9	0.736799	0.000965118	0.262236	7253	13
2.1	0.665274	0.000733084	0.333993	13641	1
2.3	0.589785	0.00091031	0.409304	20872	0
2.5	0.503183	0.000500751	0.496316	27958	0
2.7	0.421322	0.00041116	0.578267	34050	0
2.9	0.346474	0.000386488	0.65314	38811	0
3.1	0.278869	0.000141426	0.72099	42425	0
3.3	0.222193	0.000155659	0.777652	44970	0
3.5	0.171462	0.00017127	0.828367	46710	0

Table A.63 CVE=0.0 CVF=0.5 NT=5

MF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
4.0	0.980796	0.00112966	0.0180746	15934	8
4.2	0.936753	0.00209911	0.0611481	32871	1
4.4	0.828187	0.00256947	0.169244	43978	0
4.6	0.640832	0.00162031	0.357548	48139	0
4.8	0.432562	0.00117319	0.566265	49438	0
5.0	0.259608	0.000501716	0.73989	49829	0
5.2	0.137881	0.000160141	0.861959	49956	0
5.4	0.0640579	2.00056e-05	0.935922	49986	0
5.6	0.0260005	0	0.973999	49999	0
5.8	0.00888	0	0.99112	50000	0
6.0	0.00274	0	0.99726	50000	0

Table A.64 CVE=0.5 CVF=0.0 NT=10

MF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
4.0	0.0232775	0.0257882	0.950934	27881	11451
4.2	0.0134304	0.0209526	0.965617	32836	8273
4.4	0.00909706	0.0178963	0.973007	36935	5925
4.6	0.00579314	0.0144704	0.979736	40220	4114
4.8	0.00340938	0.0116526	0.984938	42823	2813
5.0	0.00245717	0.00938191	0.988161	44767	1866
5.2	0.0016	0.00771892	0.990681	46250	1187
5.4	0.000697232	0.00551447	0.993788	47330	714
5.6	0.000560294	0.00408807	0.995352	48189	431
5.8	0.000430504	0.00328003	0.996289	48780	264
6.0	0.000101665	0.00223664	0.997662	49181	158

Table A.65 CVE=2.0 CVF=0.0 NT=10

MF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
4.0	0.819461	0.010551	0.169988	853	1284
4.2	0.797069	0.00620068	0.196731	1774	717
4.4	0.74946	0.00802222	0.242518	3241	405
4.6	0.705915	0.00799851	0.286086	5376	231
4.8	0.658423	0.00563173	0.335945	8168	138
5.0	0.610982	0.00586156	0.383157	11601	88
5.2	0.567849	0.00563253	0.426518	15446	46
5.4	0.52113	0.0055499	0.47332	19640	23
5.6	0.471319	0.00383255	0.524848	23744	11
5.8	0.423114	0.00309341	0.573792	27801	7
6.0	0.379212	0.00287402	0.617914	31663	5

Table A.66 CVE=0.0 CVF=0.5 NT=10

CVF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	0.0888026	0.000220106	0.910977	49976	0
0.1	0.116089	0.000380487	0.883531	49936	0
0.2	0.168374	0.00101198	0.830614	49408	0
0.3	0.207163	0.00239189	0.790445	47243	4
0.4	0.227897	0.00366156	0.768441	43151	63
0.5	0.227579	0.00502763	0.767394	38189	384

Table A.67 CVE=0.5 MF=2.5 NT=5

CVF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	0.0159717	0.00562007	0.978408	48398	78
0.1	0.0212138	0.00584418	0.972942	48082	127
0.2	0.0339297	0.00833883	0.957731	47009	268
0.3	0.0555568	0.0124055	0.932038	44819	679
0.4	0.0752704	0.0163558	0.908374	41331	1601
0.5	0.0884241	0.0189364	0.892639	36913	3425

Table A.68 CVE=1.0 MF=2.5 NT=5

CVF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	0.00351325	0.0049454	0.991541	44688	2351
0.1	0.00430025	0.00632655	0.989373	44416	2575
0.2	0.00468656	0.00740105	0.987912	43102	3487
0.3	0.00798232	0.00835073	0.983667	40715	5054
0.4	0.0112863	0.010412	0.978302	37745	7208
0.5	0.0139937	0.0128399	0.973166	33801	10304

Table A.69 CVE=2.0 MF=2.5 NT=5

CVF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	0.259608	0.000501716	0.73989	49829	0
0.1	0.273233	0.000785673	0.725982	49639	0
0.2	0.302482	0.00182743	0.695691	48155	0
0.3	0.328988	0.0028529	0.668159	43114	18
0.4	0.337907	0.0045164	0.657577	35205	321
0.5	0.332012	0.00533294	0.662655	27002	1975

Table A.70 CVE=0.5 MF=5.0 NT=10

CVF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	0.0157913	0.0103114	0.973897	47811	49
0.1	0.0196768	0.0125813	0.967742	47213	78
0.2	0.0373038	0.0162696	0.946427	45545	248
0.3	0.0645285	0.0236173	0.911854	41749	867
0.4	0.0940077	0.0280616	0.877931	35529	2919
0.5	0.112434	0.036298	0.851268	27963	7331

Table A.71 CVE=1.0 MF=5.0 NT=10

CVF	Only EQF	Only RTC	EQF & RTC	Solutions Found	Util > 1.0
0.0	0.00245717	0.00938191	0.988161	44767	1866
0.1	0.00310243	0.00930728	0.98759	44159	2108
0.2	0.00386	0.0118642	0.984276	42228	3257
0.3	0.00678815	0.0166994	0.976512	38744	5667
0.4	0.00940176	0.0217079	0.96889	33398	9835
0.5	0.0138288	0.0257194	0.960452	26828	15867

Table A.72 CVE=2.0 MF=5.0 NT=10

CVE	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	NaN	NaN	NaN	0	0
0.25	0.03198	4e-05	0.96798	50000	0
0.50	0.0338797	0.000180104	0.96594	49971	0
0.75	0.0390764	0.00307593	0.957848	49416	4
1.00	0.0382382	0.00576673	0.955995	48381	84
1.25	0.0339425	0.00716588	0.958892	47168	421
1.50	0.0300156	0.007314	0.96267	46076	970
1.75	0.0263111	0.00571214	0.967977	45342	1681
2.00	0.0209538	0.00463897	0.974407	44622	2438

Table A.73 CVF=0.0 MF=2.5 NT=5

CVE	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	0.00489904	0.00347339	0.991628	38579	0
0.25	0.151436	0.0118439	0.83672	46944	0
0.50	0.137603	0.00292685	0.85947	48858	1
0.75	0.102197	0.00650628	0.891297	47800	62
1.00	0.0817989	0.00920643	0.908995	46272	403
1.25	0.0614541	0.0105172	0.928029	44879	1216
1.50	0.0516088	0.00850215	0.939889	43636	2121
1.75	0.0434854	0.00720095	0.949314	42911	3108
2.00	0.0362027	0.00567886	0.958118	42262	3939

Table A.74 CVF=0.25 MF=2.5 NT=5

CVE	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	0.00435901	0.0015721	0.994069	27988	0
0.25	0.180124	0.00900338	0.810872	33209	30
0.50	0.19645	0.00884864	0.794701	38763	357
0.75	0.164038	0.015333	0.820629	38479	1508
1.00	0.139555	0.0170091	0.843436	37333	3229
1.25	0.116206	0.0192525	0.864542	36203	5284
1.50	0.0945651	0.014531	0.890904	35235	7123
1.75	0.0818964	0.0131278	0.904976	34507	8745
2.00	0.0673489	0.0103794	0.922272	34106	10024

Table A.75 CVF=0.5 MF=2.5 NT=3

CVE	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	NaN	NaN	NaN	0	0
0.25	0.360921	0.0108137	0.628265	49382	0
0.50	0.294322	0.000562181	0.705116	49806	0
0.75	0.224742	0.00330745	0.77195	48678	0
1.00	0.132579	0.00677624	0.860645	47519	50
1.25	0.0862753	0.00838398	0.905341	46398	269
1.50	0.061284	0.00932202	0.929394	45591	755
1.75	0.0437098	0.00785445	0.948436	45070	1225
2.00	0.0360584	0.00634878	0.957593	44733	1904

Table A.76 CVF=0.0 MF=5.0 NT=10

CVE	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	0.0135535	0.00946014	0.976986	21987	0
0.25	0.43269	0.0158645	0.551445	39333	0
0.50	0.459562	0.00167865	0.53876	46466	1
0.75	0.350691	0.00584485	0.643464	45168	68
1.00	0.234627	0.00965035	0.755723	43729	420
1.25	0.162143	0.0109405	0.826916	42777	1196
1.50	0.117132	0.0115553	0.871313	41799	2207
1.75	0.0866704	0.0092671	0.904063	41329	3171
2.00	0.0660886	0.00896531	0.924946	40824	4130

Table A.77 CVF=0.0 MF=5.0 NT=10

CVE	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	0.00838303	0.00328407	0.988333	11571	88
0.25	0.403064	0.00442457	0.592512	21019	438
0.50	0.472742	0.00266638	0.524592	27753	1781
0.75	0.410618	0.00831953	0.581063	28367	4201
1.00	0.332089	0.0149625	0.652948	28137	6920
1.25	0.271118	0.0158908	0.712991	27689	9643
1.50	0.215721	0.0158591	0.76842	27429	12078
1.75	0.181228	0.0145129	0.804259	27286	13909
2.00	0.154212	0.0118824	0.833905	27099	15269

Table A.78 CVF=0.0 MF=5.0 NT=10

NT	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0	0	1	50000	0
3	0	0	1	50000	0
4	2e-05	0	0.99998	50000	0
5	0.0331939	0.000280118	0.966526	49979	0
6	0.671044	0.0110579	0.317898	45307	1

Table A.79 CVE=0.5 CVF=0.0 MF=2.5

NT	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0	0	1	50000	0
3	0	0	1	49996	0
4	0.00226883	0.00171695	0.996014	48924	362
5	0.0210446	0.00447758	0.974478	44667	2351
6	0.108856	0.00958159	0.881563	35276	7656

Table A.80 CVE=2.0 CVF=0.0 MF=2.5

NT	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
1	0	0	1	50000	0
2	0	4.00328e-05	0.99996	49959	0
3	0.000142025	2.02893e-05	0.999838	49287	0
4	0.00128811	0.000338975	0.998373	44251	0
5	0.00435901	0.0015721	0.994069	27988	0
6	0.0106494	0.00451441	0.984836	8639	26

Table A.81 CVE=0.0 CVF=0.5 MF=2.5

NT	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
7	0	0	1	50000	0
8	8e-05	0	0.99992	50000	0
9	0.02034	6e-05	0.9796	50000	0
10	0.294639	0.000461597	0.704899	49827	0
11	0.82283	0.00245832	0.174711	46780	0
12	0.996741	0.000352314	0.00290659	22707	2

Table A.82 CVE=0.5 CVF=0.0 MF=5.0

NT	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
7	0.000481126	0.000300704	0.999218	49883	11
8	0.00222952	0.00158093	0.99619	49338	148
9	0.0118583	0.00388317	0.984259	47899	577
10	0.0369616	0.00718635	0.955852	44668	1866
11	0.107512	0.00988737	0.8826	39242	4341
12	0.26507	0.0103777	0.724552	30739	8694

Table A.83 CVE=2.0 CVF=0.0 MF=5.0

NT	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
7	0.000820663	0.000323946	0.998855	46304	0
8	0.00238571	0.000933537	0.996681	38563	2
9	0.00460628	0.00202518	0.993369	25183	6
10	0.00838303	0.00328407	0.988333	11571	88
11	0.0127021	0.00404157	0.983256	3464	422
12	0.0214724	0.00613497	0.972393	652	1632

Table A.84 CVE=0.0 CVF=0.5 MF=5.0

MF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
1.5	0.969631	0.010846	0.0195228	461	450
1.7	0.912687	0.0162997	0.0710132	10491	54
1.9	0.75388	0.0220849	0.224035	36405	3
2.1	0.447603	0.00880701	0.54359	48257	0
2.3	0.162488	0.00156567	0.835946	49819	0
2.5	0.0331939	0.000280118	0.966526	49979	0
2.7	0.00366	2e-05	0.99632	50000	0
2.9	0.0002	0	0.9998	50000	0
3.1	2e-05	0	0.99998	50000	0
3.3	0	0	1	50000	0
3.5	0	0	1	50000	0

Table A.85 CVE=0.5 CVF=0.0 NT=5

MF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
1.5	0.375107	0.0107519	0.614141	14044	23188
1.7	0.23333	0.0132041	0.753466	22796	16157
1.9	0.141394	0.0113478	0.847259	30843	10519
2.1	0.0811137	0.00854815	0.910338	37318	6612
2.3	0.0410486	0.00639812	0.952553	41731	4091
2.5	0.0210446	0.00447758	0.974478	44667	2351
2.7	0.0121797	0.00383158	0.983989	46717	1229
2.9	0.00615589	0.00259962	0.991244	48084	586
3.1	0.00296269	0.00161416	0.995423	48942	261
3.3	0.0017386	0.000909734	0.997352	49465	113
3.5	0.000623304	0.000522771	0.998854	49735	43

Table A.86 CVE=2.0 CVF=0.0 NT=5

MF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
1.5	0.0258152	0.00543478	0.96875	736	161
1.7	0.015795	0.00631801	0.977887	2849	39
1.9	0.010729	0.00330124	0.98597	7270	13
2.1	0.00841936	0.00204993	0.989531	13659	1
2.3	0.00531024	0.002392	0.992298	20903	0
2.5	0.00435901	0.0015721	0.994069	27988	0
2.7	0.00372685	0.00120316	0.99507	34077	0
2.9	0.00242056	0.000978524	0.996601	38834	0
3.1	0.00155499	0.000589011	0.997856	42444	0
3.3	0.000955917	0.000444612	0.998599	44983	0
3.5	0.000834867	0.000256882	0.998908	46714	0

Table A.87 CVE=0.0 CVF=0.5 NT=5

MF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
4.0	0.99617	0.000627904	0.00320231	15926	8
4.2	0.969825	0.00222053	0.0279544	32875	1
4.4	0.878648	0.00372482	0.117627	44029	0
4.6	0.706799	0.00251131	0.290689	48182	0
4.8	0.492506	0.00123379	0.50626	49441	0
5.0	0.294639	0.000461597	0.704899	49827	0
5.2	0.14493	0.000140126	0.854929	49955	0
5.4	0.0611971	2.00056e-05	0.938783	49986	0
5.6	0.0210604	0	0.97894	49999	0
5.8	0.00602	0	0.99398	50000	0
6.0	0.00122	0	0.99878	50000	0

Table A.88 CVE=0.5 CVF=0.0 NT=10

MF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
4.0	0.290296	0.0101673	0.699537	27441	11451
4.2	0.200832	0.00900123	0.790166	32440	8273
4.4	0.13292	0.00974584	0.857334	36631	5925
4.6	0.086627	0.00845507	0.904918	39976	4114
4.8	0.0565047	0.0080855	0.93541	42669	2813
5.0	0.0369616	0.00718635	0.955852	44668	1866
5.2	0.0247146	0.00593498	0.96935	46167	1187
5.4	0.0156058	0.00467329	0.979721	47290	714
5.6	0.0101973	0.00328141	0.986521	48150	431
5.8	0.00629744	0.00266667	0.991036	48750	264
6.0	0.00404727	0.00199313	0.99396	49169	158

Table A.89 CVE=2.0 CVF=0.0 NT=10

MF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
4.0	0.0117647	0.00705882	0.981176	850	1284
4.2	0.016283	0.0101067	0.97361	1781	717
4.4	0.00958269	0.00618238	0.984235	3235	405
4.6	0.0117297	0.00707503	0.981195	5371	231
4.8	0.0101766	0.00416871	0.985655	8156	138
5.0	0.00838303	0.00328407	0.988333	11571	88
5.2	0.00746899	0.00246801	0.990063	15397	46
5.4	0.00643678	0.00224777	0.991315	19575	23
5.6	0.00502279	0.00164613	0.993331	23692	11
5.8	0.00540307	0.00169296	0.992904	27762	7
6.0	0.00414373	0.00132853	0.994528	31614	5

Table A.90 CVE=0.0 CVF=0.5 NT=10

CVF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	0.0331939	0.000280118	0.966526	49979	0
0.1	0.0557858	0.000480567	0.943734	49941	0
0.2	0.116325	0.00216315	0.881512	49465	0
0.3	0.170057	0.0043729	0.82557	47337	4
0.4	0.197128	0.0057353	0.797137	43241	63
0.5	0.199108	0.00910134	0.791791	38346	384

Table A.91 CVE=0.5 MF=2.5 NT=5

CVF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	0.0378505	0.0056817	0.956468	48401	78
0.1	0.0476022	0.00592688	0.946471	48086	127
0.2	0.0689552	0.00848647	0.922558	47016	268
0.3	0.1009	0.0112583	0.887841	44767	679
0.4	0.123406	0.0160701	0.860524	41319	1601
0.5	0.141044	0.0181118	0.840844	36882	3425

Table A.92 CVE=1.0 MF=2.5 NT=5

CVF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	0.0210446	0.00447758	0.974478	44667	2351
0.1	0.0252817	0.0055205	0.969198	44380	2575
0.2	0.0326708	0.00657131	0.960758	43066	3487
0.3	0.0437326	0.00747314	0.948794	40679	5054
0.4	0.0580834	0.00844175	0.933475	37670	7208
0.5	0.0702108	0.0106739	0.919115	33727	10304

Table A.93 CVE=2.0 MF=2.5 NT=5

CVF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	0.294639	0.000461597	0.704899	49827	0
0.1	0.346969	0.00100705	0.652024	49650	0
0.2	0.439062	0.00116368	0.559774	48123	0
0.3	0.489396	0.0013473	0.509257	43049	18
0.4	0.498163	0.00173755	0.5001	35107	321
0.5	0.475526	0.00252544	0.521949	26926	1975

Table A.94 CVE=0.5 MF=5.0 NT=10

CVF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	0.137392	0.00700915	0.855599	47652	49
0.1	0.159256	0.00823299	0.832511	47006	78
0.2	0.215175	0.0088707	0.775954	45205	248
0.3	0.271862	0.0101022	0.718036	41179	867
0.4	0.318566	0.0110828	0.670351	34919	2919
0.5	0.338915	0.0146623	0.646422	27349	7331

Table A.95 CVE=1.0 MF=5.0 NT=10

CVF	Only EQF	Only LM	EQF & LM	Solutions Found	Util > 1.0
0.0	0.0369616	0.00718635	0.955852	44668	1866
0.1	0.043034	0.00651754	0.950449	44035	2108
0.2	0.0610585	0.00787008	0.931071	42058	3257
0.3	0.0917832	0.0100047	0.898212	38482	5667
0.4	0.122571	0.0111676	0.866261	33042	9835
0.5	0.157758	0.0125798	0.829663	26471	15867

Table A.96 CVE=2.0 MF=5.0 NT=10

END

1 3 0 6 9 7

FIN