소프트웨어 특강 1 final project

조대호 교수님

2017314804

정희수

**Contents**

-table

-3 diagrams for pl, psm, pd, pc atomic models

-tests of atomic models

-20 log files

-14 model program files

-SES source codes

-SES diagram and screen copy

Table

1. Theoretical

|  |  |  |  |
| --- | --- | --- | --- |
| Architecture | Parameters | Turnaround Time | Throughput |
| Pl | P(6~8) | P(6~8) | 1/p |
| Multiserver | P1,p2,p3(2~4) | 3/throughput | 1/p1+1/p2+1/p3 |
| Divide & conquer | P1,p2,p3 cp cm(2~4) | Cp+cm+max(pi) | 1/max(pi,cp,cm) |

1. simulation cases

|  |  |  |
| --- | --- | --- |
| Architecture | Turnaround Time | Throughput |
| Pl | 7.600932 | 0.113475 |
| Multiserver | 7.454305 | 0.318953 |
| Divide&Conquer | 8.01111 | 0.101075 |

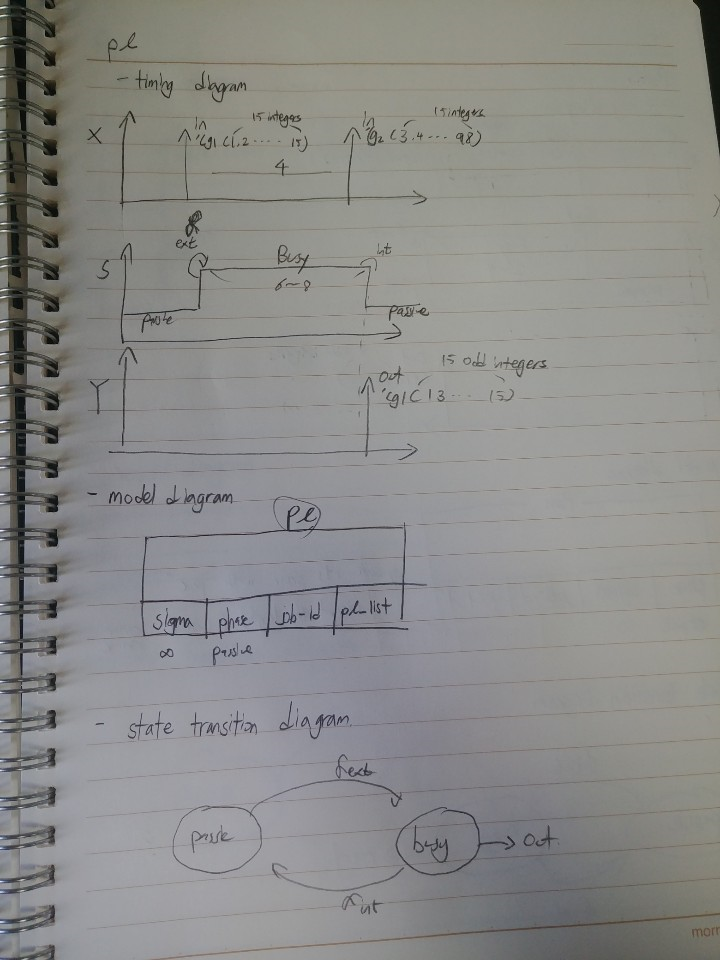
이상적인 결과는 turnaround time의 경우 pl과 multiserver는 같고 multiserver와 divide&conquere은 비슷해야하고 throughput의 경우 pl은 8분의 1에서 6분의 1사이, multiserver는 pl의 3배정도 나와야 하고 divide&conquer의 경우는 걸리는 시간의 최댓값이 p1,p2,p3이므로 pl과 비슷한 값이 나와야한다. 결과는 좋게 나온 것 같다.

Discussions:

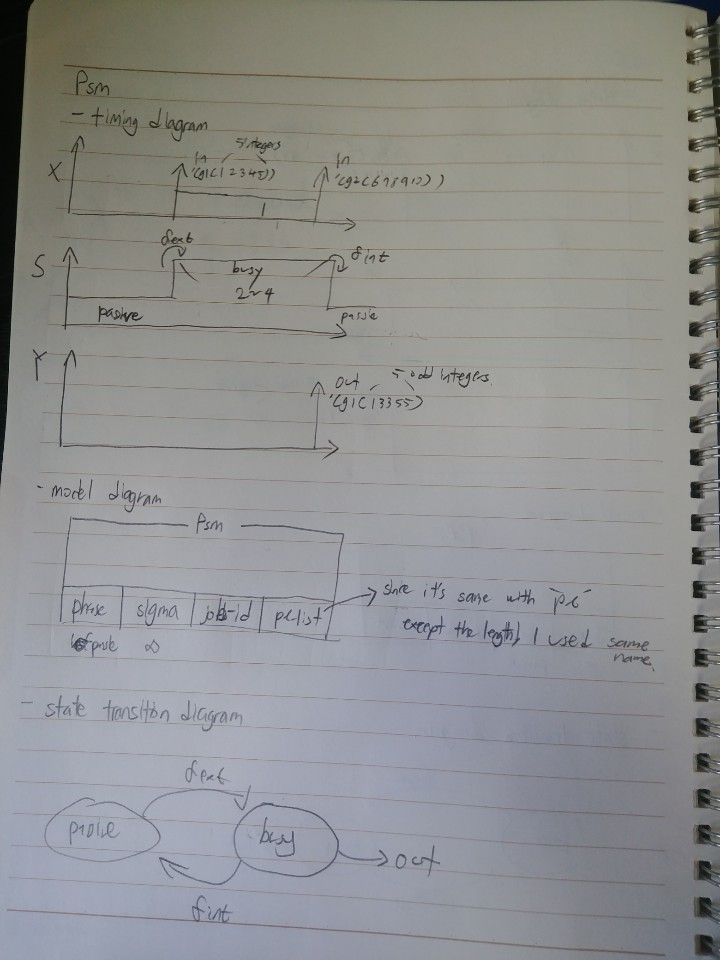
Input 이 generator에 의해 3초마다 들어온다. 하지만 예를 들어 pl의 경우 한번 pl이 끝나는데 6초에서 10초가 걸린다 즉 input이 기다리는 경우가 생긴다. 이러한 경우 이론적으로 구한 결과보다 input을 더 늦게 받는 경향이 생겨서 turnaround time이 더 늘어난다. 이와 별개로 또한 multiserver의 경우 이상적인 경우는 p1,p2,p3의 process time이 pl의 processing time의 정확히 3분의 1이다. 하지만 이번 과제의 경우 각각 p1,p2,p3의 process time 이 2~4사이이고 pl의 process time이 6~10 이기 때문에 정확히 3배가 아닐 확률이 높기 때문에 turnaround time 과 throughput에서 더 안좋은 결과가 나왔다.

-3 diagrams for atomic models.

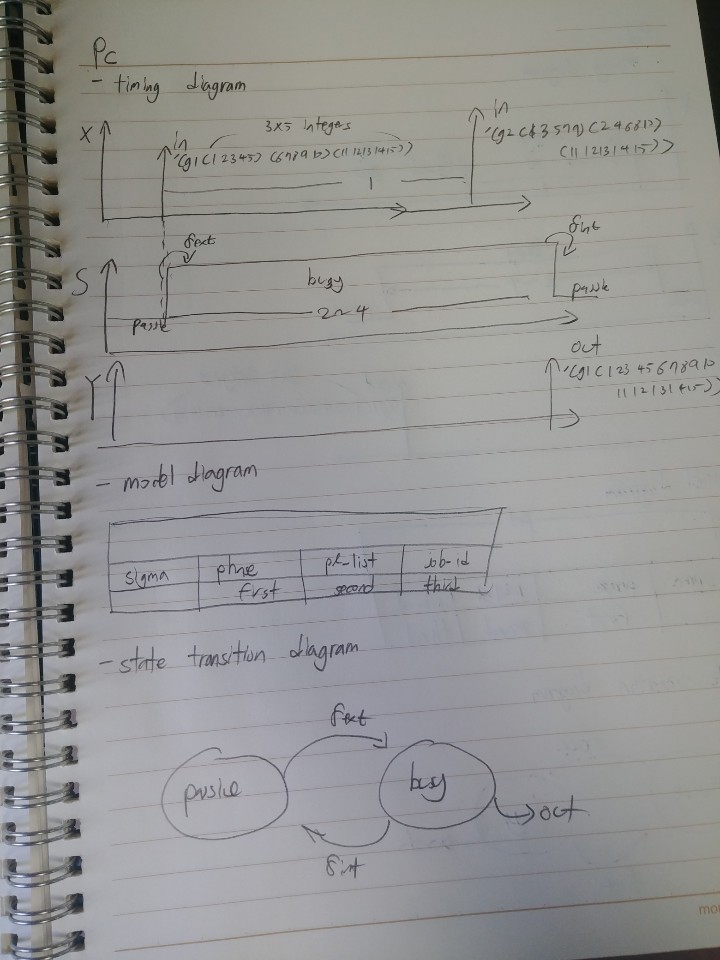
- pl model



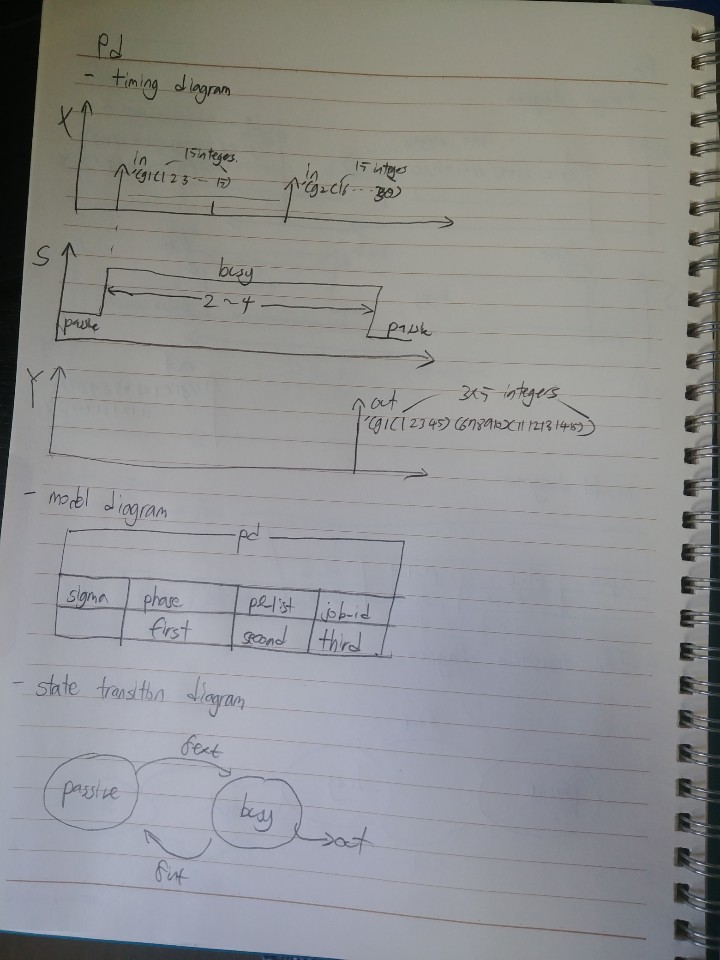
- psm



-pc



-pd



-tests

OK

[2] (load "mbase/psm.m")

Model of type atomic-models with name PSM made.

Processor of type simulators with name S:PSM made.

OK

[3] (load "mbase/pl.m")

Model of type atomic-models with name PL made.

Processor of type simulators with name S:PL made.

OK

[4] (load "mbase/pc.m")

Model of type atomic-models with name PC made.

Processor of type simulators with name S:PC made.

OK

[5] (load "mbase/pd.m")

Model of type atomic-models with name PD made.

Processor of type simulators with name S:PD made.

OK

[6] (send pl inject 'in '(g1 (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15))3)

state s =

state s = (6 BUSY (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15) G1)state s = ()()

[7] (send pl output?)

output y = output y = OUT (G1 (1 3 3 5 5 7 7 9 9

11 11 13 13 15 15))#(((|#!STRUCTURE| . CONTENT)) OUT (G1 (1 3 3 5 5 7 7 9 9 11 11 13 13 15 15)))

[8] (send pl int-transition)

state s =

state s = (INF PASSIVE (1 3 3 5 5 7 7 9 9 11 11 13 13 15 15) G1)state s = ()()

[9] (send psm inject 'in '(g1 (1 2 3 4 5) )4)

state s =

state s = (2 BUSY (1 2 3 4 5) G1)state s = ()()

[10] (send psm output?)

output y = output y = OUT (G1 (1 3 3 5 5))#(((|#!STRUCTURE| . CONTENT)) OUT (G1 (1 3 3 5 5)))

[11] (send psm int-transition)

state s =

state s = (INF PASSIVE (1 3 3 5 5) G1)state s = ()()

[12] (send pd inject 'in '(g1 (1 1 1 1 1 2 2 2 2 2 3 3 3 3 3))3)

state s =

state s = (3 BUSY (1 1 1 1 1 2 2 2 2 2 3 3 3 3 3) G1 - - -)state s = ()()

[13] (send pd output?)

output y = output y = OUT (G1 (1 1 1 1 1) (2 2 2

2 2) (3 3 3 3 3))#(((|#!STRUCTURE| . CONTENT)) OUT (G1 (1 1 1 1 1) (2 2 2 2 2) (3 3 3 3 3)))

[14] (send pd int-transition)

state s =

state s = (INF PASSIVE - G1 (1 1 1 1 1) (2 2 2 2 2) (3 3 3 3 3))state s = ()()

[15] (send pc inject 'in '(g1 (1 1 1 1 1) (2 2 2 2 2) (3 3 3 3 3))3)

state s =

state s = (3 BUSY - G1 (1 1 1 1 1) (2 2 2 2 2) (3 3 3 3 3))state s = ()()

[16] (send pc output?)

output y = output y = OUT (G1 (1 1 1 1 1 2 2 2 2

2 3 3 3 3 3))#(((|#!STRUCTURE| . CONTENT)) OUT (G1 (1 1 1 1 1 2 2 2 2 2 3 3 3 3 3)))

[17] (send pc int-transition)

state s =

state s = (INF PASSIVE (1 1 1 1 1 2 2 2 2 2 3 3 3 3 3) G1 (1 1 1 1 1) (2 2 2

2 2) (3 3 3 3 3))state s = ()()

[18] (transcript-off)

-20 log files

Ef-pl.m

The arrived list: ((G57 99) (G56 96) (G55 93) (G54 90) (G53 87) (G52 84) (G51 81

) (G50 78) (G49 75) (G48 72) (G47 69) (G46 66) (G45 63) (G44 60) (G43 57) (G42

54) (G41 51) (G40 48) (G39 45) (G38 42) (G37 39) (G36 36) (G35 33) (G34 30) (G33

27) (G32 24) (G31 21) (G30 18) (G29 15) (G28 12) (G27 9) (G26 6) (G25 3) (G24 0)

)

The solved list: (G51 G50 G46 G42 G39 G38 G34 G31 G28 G27)

Avg. turnaround time: 8

ThruPut: 0.0989010989010989

The arrived list: ((G41 99) (G40 96) (G39 93) (G38 90) (G37 87) (G36 84) (G35 81

) (G34 78) (G33 75) (G32 72) (G31 69) (G30 66) (G29 63) (G28 60) (G27 57) (G26

54) (G25 51) (G24 48) (G23 45) (G22 42) (G21 39) (G20 36) (G19 33) (G18 30) (G17

27) (G16 24) (G15 21) (G14 18) (G13 15) (G12 12) (G11 9) (G10 6) (G9 3) (G8 0))

The solved list: (G39 G37 G34 G32 G30 G27 G24 G21 G19 G16 G13 G10 G8)

Avg. turnaround time: 7.30769230769231

ThruPut: 0.129460580912863

The arrived list: ((G41 99) (G40 96) (G39 93) (G38 90) (G37 87) (G36 84) (G35 81

) (G34 78) (G33 75) (G32 72) (G31 69) (G30 66) (G29 63) (G28 60) (G27 57) (G26

54) (G25 51) (G24 48) (G23 45) (G22 42) (G21 39) (G20 36) (G19 33) (G18 30) (G17

27) (G16 24) (G15 21) (G14 18) (G13 15) (G12 12) (G11 9) (G10 6) (G9 3) (G8 0))

The solved list: (G36 G33 G31 G28 G26 G23 G20 G17 G14 G11 G8)

Avg. turnaround time: 7.54545454545455

ThruPut: 0.10934393638171

The arrived list: ((G41 99) (G40 96) (G39 93) (G38 90) (G37 87) (G36 84) (G35 81

) (G34 78) (G33 75) (G32 72) (G31 69) (G30 66) (G29 63) (G28 60) (G27 57) (G26

54) (G25 51) (G24 48) (G23 45) (G22 42) (G21 39) (G20 36) (G19 33) (G18 30) (G17

27) (G16 24) (G15 21) (G14 18) (G13 15) (G12 12) (G11 9) (G10 6) (G9 3) (G8 0))

The solved list: (G38 G36 G33 G30 G27 G24 G22 G19 G16 G13 G11 G8)

Avg. turnaround time: 7.33333333333333

ThruPut: 0.12

The arrived list: ((G43 99) (G42 96) (G41 93) (G40 90) (G39 87) (G38 84) (G37 81

) (G36 78) (G35 75) (G34 72) (G33 69) (G32 66) (G31 63) (G30 60) (G29 57) (G28

54) (G27 51) (G26 48) (G25 45) (G24 42) (G23 39) (G22 36) (G21 33) (G20 30) (G19

27) (G18 24) (G17 21) (G16 18) (G15 15) (G14 12) (G13 9) (G12 6) (G11 3) (G10 0)

)

The solved list: (G39 G36 G33 G30 G27 G25 G22 G19 G16 G13 G10)

Avg. turnaround time: 7.81818181818182

ThruPut: 0.109670987038883

Ef-mul.m

The arrived list: ((G49 99) (G48 96) (G47 93) (G46 90) (G45 87) (G44 84) (G43 81

) (G42 78) (G41 75) (G40 72) (G39 69) (G38 66) (G37 63) (G36 60) (G35 57) (G34

54) (G33 51) (G32 48) (G31 45) (G30 42) (G29 39) (G28 36) (G27 33) (G26 30) (G25

27) (G24 24) (G23 21) (G22 18) (G21 15) (G20 12) (G19 9) (G18 6) (G17 3) (G16 0)

)

The solved list: (G46 G45 G44 G43 G42 G41 G40 G38 G37 G36 G35 G34 G33 G32 G31

G30 G29 G28 G27 G26 G25 G24 G23 G22 G21 G20 G19 G18 G17 G16)

Avg. turnaround time: 7.43333333333333

ThruPut: 0.316709137240626

The arrived list: ((G199 99) (G198 96) (G197 93) (G196 90) (G195 87) (G194 84) (

G193 81) (G192 78) (G191 75) (G190 72) (G189 69) (G188 66) (G187 63) (G186 60) (

G185 57) (G184 54) (G183 51) (G182 48) (G181 45) (G180 42) (G179 39) (G178 36) (

G177 33) (G176 30) (G175 27) (G174 24) (G173 21) (G172 18) (G171 15) (G170 12) (

G169 9) (G168 6) (G167 3) (G166 0))

The solved list: (G197 G196 G195 G194 G193 G192 G191 G190 G189 G188 G187 G186

G185 G184 G183 G182 G181 G180 G179 G178 G177 G175 G174 G173 G172 G171 G170 G169

G168 G167 G166)

Avg. turnaround time: 7.38709677419355

ThruPut: 0.327464788732394

The arrived list: ((G349 99) (G348 96) (G347 93) (G346 90) (G345 87) (G344 84) (

G343 81) (G342 78) (G341 75) (G340 72) (G339 69) (G338 66) (G337 63) (G336 60) (

G335 57) (G334 54) (G333 51) (G332 48) (G331 45) (G330 42) (G329 39) (G328 36) (

G327 33) (G326 30) (G325 27) (G324 24) (G323 21) (G322 18) (G321 15) (G320 12) (

G319 9) (G318 6) (G317 3) (G316 0))

The solved list: (G347 G344 G343 G342 G341 G340 G339 G338 G337 G336 G334 G333

G332 G331 G330 G329 G328 G327 G326 G325 G324 G323 G322 G321 G320 G319 G318 G317

G316)

Avg. turnaround time: 7.75862068965517

ThruPut: 0.306878306878307

The arrived list: ((G49 99) (G48 96) (G47 93) (G46 90) (G45 87) (G44 84) (G43 81

) (G42 78) (G41 75) (G40 72) (G39 69) (G38 66) (G37 63) (G36 60) (G35 57) (G34

54) (G33 51) (G32 48) (G31 45) (G30 42) (G29 39) (G28 36) (G27 33) (G26 30) (G25

27) (G24 24) (G23 21) (G22 18) (G21 15) (G20 12) (G19 9) (G18 6) (G17 3) (G16 0)

)

The solved list: (G47 G46 G45 G44 G43 G42 G41 G40 G39 G38 G37 G36 G35 G34 G33

G32 G31 G30 G29 G28 G27 G26 G25 G24 G23 G22 G21 G19 G18 G17 G16)

Avg. turnaround time: 7.2258064516129

ThruPut: 0.32688927943761

The arrived list: ((G199 99) (G198 96) (G197 93) (G196 90) (G195 87) (G194 84) (

G193 81) (G192 78) (G191 75) (G190 72) (G189 69) (G188 66) (G187 63) (G186 60) (

G185 57) (G184 54) (G183 51) (G182 48) (G181 45) (G180 42) (G179 39) (G178 36) (

G177 33) (G176 30) (G175 27) (G174 24) (G173 21) (G172 18) (G171 15) (G170 12) (

G169 9) (G168 6) (G167 3) (G166 0))

The solved list: (G196 G195 G194 G193 G192 G191 G190 G189 G188 G187 G186 G185

G184 G183 G182 G181 G180 G179 G178 G177 G176 G175 G174 G173 G172 G171 G169 G168

G167 G166)

Avg. turnaround time: 7.46666666666667

ThruPut: 0.316824471959213

Ef-dc.m

The arrived list: ((G115 99) (G114 96) (G113 93) (G112 90) (G111 87) (G110 84) (

G109 81) (G108 78) (G107 75) (G106 72) (G105 69) (G104 66) (G103 63) (G102 60) (

G101 57) (G100 54) (G99 51) (G98 48) (G97 45) (G96 42) (G95 39) (G94 36) (G93 33

) (G92 30) (G91 27) (G90 24) (G89 21) (G88 18) (G87 15) (G86 12) (G85 9) (G84 6)

(G83 3) (G82 0))

The solved list: (G107 G106 G100 G96 G94 G93 G88 G83)

Avg. turnaround time: 8

ThruPut: 0.0769230769230769

The arrived list: ((G173 99) (G172 96) (G171 93) (G170 90) (G169 87) (G168 84) (

G167 81) (G166 78) (G165 75) (G164 72) (G163 69) (G162 66) (G161 63) (G160 60) (

G159 57) (G158 54) (G157 51) (G156 48) (G155 45) (G154 42) (G153 39) (G152 36) (

G151 33) (G150 30) (G149 27) (G148 24) (G147 21) (G146 18) (G145 15) (G144 12) (

G143 9) (G142 6) (G141 3) (G140 0))

The solved list: (G170 G167 G166 G162 G155 G151 G147 G142 G141)

Avg. turnaround time: 8.55555555555556

ThruPut: 0.0884520884520885

The arrived list: ((G57 99) (G56 96) (G55 93) (G54 90) (G53 87) (G52 84) (G51 81

) (G50 78) (G49 75) (G48 72) (G47 69) (G46 66) (G45 63) (G44 60) (G43 57) (G42

54) (G41 51) (G40 48) (G39 45) (G38 42) (G37 39) (G36 36) (G35 33) (G34 30) (G33

27) (G32 24) (G31 21) (G30 18) (G29 15) (G28 12) (G27 9) (G26 6) (G25 3) (G24 0)

)

The solved list: (G51 G50 G48 G45 G41 G38 G37 G33 G32 G28 G24)

Avg. turnaround time: 8.18181818181818

ThruPut: 0.11011011011011

The arrived list: ((G57 99) (G56 96) (G55 93) (G54 90) (G53 87) (G52 84) (G51 81

) (G50 78) (G49 75) (G48 72) (G47 69) (G46 66) (G45 63) (G44 60) (G43 57) (G42

54) (G41 51) (G40 48) (G39 45) (G38 42) (G37 39) (G36 36) (G35 33) (G34 30) (G33

27) (G32 24) (G31 21) (G30 18) (G29 15) (G28 12) (G27 9) (G26 6) (G25 3) (G24 0)

)

The solved list: (G54 G53 G49 G46 G43 G42 G39 G37 G33 G32 G24)

Avg. turnaround time: 7.81818181818182

ThruPut: 0.109670987038883

The arrived list: ((G57 99) (G56 96) (G55 93) (G54 90) (G53 87) (G52 84) (G51 81

) (G50 78) (G49 75) (G48 72) (G47 69) (G46 66) (G45 63) (G44 60) (G43 57) (G42

54) (G41 51) (G40 48) (G39 45) (G38 42) (G37 39) (G36 36) (G35 33) (G34 30) (G33

27) (G32 24) (G31 21) (G30 18) (G29 15) (G28 12) (G27 9) (G26 6) (G25 3) (G24 0)

)

The solved list: (G54 G53 G51 G50 G43 G40 G36 G35 G32 G31 G28 G24)

Avg. turnaround time: 7.5

ThruPut: 0.120218579234973

14 files

1. pl.m

(make-pair atomic-models 'pl)

;;;;;;;; set up additional variables job-id and processing-time

(send pl def-state

'(

;;;state-variables:

pl\_list ;

job\_id ;

)

)

;;;;;;;; initialize variables

(send pl set-s

(make-state 'sigma 'inf

'phase 'passive

'pl\_list '()

'job\_id '()

)

)

;;;;;;;; define the external transition function

(define (ex-f s e x)

(case (content-port x)

('in (case (state-phase s)

('passive (set! (state-pl\_list s) (cadr (content-value x) ))

(set! (state-job\_id s) (car (content-value x) ) )

(hold-in 'busy (truncate (unifrm 6 10 2)))

)

('busy (continue))

)

)

)

)

;;;;;;;; define the internal transition function

(define (in-f s)

(case (state-phase s)

('busy

(passivate)

)

)

)

;;;;;;;; define the output function

(define (out-f s)

(case (state-phase s)

('busy

(set! (state-pl\_list s) ( map (lambda(x) (if (= 0 (remainder x 2) ) (+ x 1) x) ) (state-pl\_list s) ))

(make-content 'port 'out 'value (append (list(state-job\_id s) ) (list(state-pl\_list s)) ))

)

(else

(continue)

)

)

)

;;;;;;;; assignment to the model

(send pl set-ext-transfn ex-f)

(send pl set-int-transfn in-f)

(send pl set-outputfn out-f)

1. psm.m

(make-pair atomic-models 'psm)

;;;;;;;; set up additional variables job-id and processing-time

(send psm def-state

'(

;;;state-variables:

pl\_list ;

job\_id ;

)

)

;;;;;;;; initialize variables

(send psm set-s

(make-state 'sigma 'inf

'phase 'passive

'pl\_list '()

'job\_id '()

)

)

;;;;;;;; define the external transition function

(define (ex-f s e x)

(case (content-port x)

('in (case (state-phase s)

('passive (set! (state-pl\_list s) (cadr (content-value x) ))

(set! (state-job\_id s) (car (content-value x) ) )

(hold-in 'busy (truncate (unifrm 2 4 2)))

)

('busy (continue))

)

)

)

)

;;;;;;;; define the internal transition function

(define (in-f s)

(case (state-phase s)

('busy

(passivate)

)

)

)

;;;;;;;; define the output function

(define (out-f s)

(case (state-phase s)

('busy

(set! (state-pl\_list s) ( map (lambda(x) (if (= 0 (remainder x 2) ) (+ x 1) x) ) (state-pl\_list s) ))

(make-content 'port 'out 'value (append (list(state-job\_id s) ) (list(state-pl\_list s) )))

)

(else

(continue)

)

)

)

;;;;;;;; assignment to the model

(send psm set-ext-transfn ex-f)

(send psm set-int-transfn in-f)

(send psm set-outputfn out-f)

1. pd.m

(make-pair atomic-models 'pd)

;;;;;;;; set up additional variables job-id and processing-time

(send pd def-state

'(

;;;state-variables:

pl\_list ;

job\_id ;

first

second

third

)

)

;;;;;;;; initialize variables

(send pd set-s

(make-state 'sigma 'inf

'phase 'passive

'pl\_list '()

'job\_id '()

'first '()

'second '()

'third '()

)

)

;;;;;;;; define the external transition function

(define (ex-f s e x)

(case (content-port x)

('in (case (state-phase s)

('passive (set! (state-pl\_list s) (cadr (content-value x) ))

(set! (state-job\_id s) (car (content-value x) ) )

(hold-in 'busy (truncate (unifrm 2 4 2)))

)

('busy (continue))

)

)

)

)

;;;;;;;; define the internal transition function

(define (in-f s)

(case (state-phase s)

('busy

(passivate)

)

)

)

;;;;;;;; define the output function

(define (out-f s)

(case (state-phase s)

('busy

(set! (state-first s) (cons (car (state-pl\_list s) ) '() ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-first s) (cons (car (state-pl\_list s) ) (state-first s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-first s) (cons (car (state-pl\_list s) ) (state-first s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-first s) (cons (car (state-pl\_list s) ) (state-first s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-first s) (cons (car (state-pl\_list s) ) (state-first s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-second s) (cons (car (state-pl\_list s) ) '() ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-second s) (cons (car (state-pl\_list s) ) (state-second s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-second s) (cons (car (state-pl\_list s) ) (state-second s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-second s) (cons (car (state-pl\_list s) ) (state-second s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-second s) (cons (car (state-pl\_list s) ) (state-second s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-third s) (cons (car (state-pl\_list s) ) '() ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-third s) (cons (car (state-pl\_list s) ) (state-third s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-third s) (cons (car (state-pl\_list s) ) (state-third s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-third s) (cons (car (state-pl\_list s) ) (state-third s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-third s) (cons (car (state-pl\_list s) ) (state-third s) ) )

(set! (state-pl\_list s) (cdr (state-pl\_list s) ))

(set! (state-first s) (reverse (state-first s) ) )

(set! (state-second s) (reverse (state-second s) ) )

(set! (state-third s) (reverse (state-third s) ) )

(make-content 'port 'out 'value (append (append (append (list(state-job\_id s)) (list(state-first s))) (list(state-second s))) (list(state-third s))) )

)

(else

(continue)

)

)

)

;;;;;;;; assignment to the model

(send pd set-ext-transfn ex-f)

(send pd set-int-transfn in-f)

(send pd set-outputfn out-f)

1. pc.m

(make-pair atomic-models 'pc)

;;;;;;;; set up additional variables job-id and processing-time

(send pc def-state

'(

;;;state-variables:

pl\_list ;

job\_id ;

first

second

third

)

)

;;;;;;;; initialize variables

(send pc set-s

(make-state 'sigma 'inf

'phase 'passive

'pl\_list '()

'job\_id '()

'first '()

'second '()

'third '()

)

)

;;;;;;;; define the external transition function

(define (ex-f s e x)

(case (content-port x)

('in (case (state-phase s)

('passive (set! (state-job\_id s) (car (content-value x) ) )

(set! (state-first s) (cadr (content-value x) ) )

(set! (state-second s) (caddr (content-value x) ) )

(set! (state-third s) (cadddr (content-value x) ) )

(hold-in 'busy (truncate (unifrm 2 4 2)))

)

('busy (continue))

)

)

)

)

;;;;;;;; define the internal transition function

(define (in-f s)

(case (state-phase s)

('busy

(passivate)

)

)

)

;;;;;;;; define the output function

(define (out-f s)

(case (state-phase s)

('busy

( set! (state-pl\_list s) (append (state-pl\_list s) (state-first s) ) )

( set! (state-pl\_list s) (append (state-pl\_list s) (state-second s) ) )

( set! (state-pl\_list s) (append (state-pl\_list s) (state-third s) ) )

(make-content 'port 'out 'value (append (list(state-job\_id s) ) (list(state-pl\_list s) )))

)

(else

(continue)

)

)

)

;;;;;;;; assignment to the model

(send pc set-ext-transfn ex-f)

(send pc set-int-transfn in-f)

(send pc set-outputfn out-f)

1. genr.m

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Generator genr.m;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;-------------------------------------------------------------

; This file contains the definition of the job generator

;-------------------------------------------------------------

; It should perform following tasks:

; generates a job every inter-arrival-time

; Option: stops the generating sequence when receiving a stop

; signal (stop)

;-------------------------------------------------------------

;; create a generator

(make-pair atomic-models 'genr)

; add another state variable: inter-arrival time

(send genr def-state '(inter-arrival-time))

; initialization

(send genr set-s (make-state 'sigma 0

'phase 'active

'inter-arrival-time 3

)

)

;; Add external transition function to terminate the generator

;; when the experiment is over instead of using keyboard interrupt

(define (ext-genr s e x)

(case (content-port x)

('stop

(passivate) ;when receive stop signal passivate

)

(else (continue))

)

)

;; definition of internal transition function

(define (int-genr s)

(case (state-phase s)

('active

(set! (state-sigma s) (state-inter-arrival-time s))

) ;;;reset sigma each time an internal transition occurs

;;; Note: not really necessary for fixed inter-arrival

;;; time

) )

;; definition of output function

; output the jobname (gensym) to port 'out

(define (out-genr s)

(case (state-phase s)

('active

(make-content 'port 'out 'value (list (gensym) (list (random 100) (random 100) (random 100) (random 100) (random 100) (random 100) (random 100) (random 100) (random 100) (random 100) (random 100) (random 100) (random 100) (random 100) (random 100) )))

)

(else (make-content)

) ) )

;; connect the definitions of functions to generator

(send genr set-int-transfn int-genr)

(send genr set-ext-transfn ext-genr)

(send genr set-outputfn out-genr)

1. trand.m

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Transducer transd.m;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;-------------------------------------------------------------

; This file contains the definition of transducer

;-------------------------------------------------------------

;

; Transducer designed for the measurement of

;

; 1. average turnaround time of processed jobs

; 2. thoughput

;

(define observation-interval 100)

; now start model definition proper

(make-pair atomic-models 'transd)

(send transd def-state '(arrived-list ;; all jobs which have arrived

solved-list ;; all jobs which have been processed

clock ;; local clock

total-ta) ;; total turnaround time of processed

;; jobs

)

(send transd set-s (make-state

'sigma observation-interval

'phase 'active

'arrived-list '()

'solved-list '()

'clock 0

'total-ta 0

) )

;; external transition function takes care of recording arriving

;; and departing jobs and of accumulating total turnaround time

(define (ext-t s e x)

(let (

(problem-id (car (content-value x)))

)

(set! (state-clock s) (+ (state-clock s) e))

(case (content-port x)

('ariv (set! (state-arrived-list s)

(cons (list problem-id (state-clock s))

(state-arrived-list s)))

)

('solved (let\* (

(pair (assoc problem-id (state-arrived-list s)))

(prob-arrival-time (cadr pair))

(turn-around-time

(when prob-arrival-time

(- (state-clock s) prob-arrival-time))

)

)

(when prob-arrival-time

(set! (state-total-ta s)

(+ (state-total-ta s) turn-around-time))

(set! (state-solved-list s)

(cons problem-id (state-solved-list s)))

)

)

)

(else (bkpt "error: invalid input port name --> " (content-port x)))

)

)

(continue)

)

;; internal transition function is called only at end of run

(define (int-t s)

(case (state-phase s)

('active (passivate))

))

;; output function serves to compute summary indexes: throughput and

;; average turnaround time

(define (out-t s)

(case (state-phase s)

('active

(let\* (

; a port log-file is opened to record transducer output in file "log"

(log-file (open-output-file "log"))

; average turn-around time: total-ta divided by number of processed jobs

(avg-ta-time

(if (NULL? (state-solved-list s))

'()

(/ (state-total-ta s) (length (state-solved-list s)))

)

)

; thruput: number of processed jobs divided by observation interval

(thruput

(if (= (state-clock s) 0)

'()

(/ (- (length (state-solved-list s)) 1)

(- (state-clock s) avg-ta-time))

)

)

)

(newline log-file)

(display "The arrived list: " log-file)

(display (state-arrived-list s) log-file)

(newline log-file)

(display "The solved list: " log-file)

(display (state-solved-list s) log-file)

(newline log-file)

(display "Avg. turnaround time: " log-file)

(display avg-ta-time log-file)

(newline log-file)

(display "ThruPut: " log-file)

(display thruput log-file)

(newline log-file)

(close-output-port log-file)

(make-content 'port 'out 'value (list avg-ta-time thruput))

) ;;let

) ;;active

(else (make-content))

))

(send transd set-ext-transfn ext-t)

(send transd set-int-transfn int-t)

(send transd set-outputfn out-t)

1. mul-c

;Multi-sever Co-ordinator

(make-pair atomic-models 'mul-c)

(send mul-c def-state '(

spfour1-s

spfour2-s

spfour3-s

out-port

job

)

)

(send mul-c set-s (make-state 'sigma 'inf

'phase 'passive

'spfour1-s 'passive

'spfour2-s 'passive

'spfour3-s 'passive

'out-port '()

'job '()

)

)

;external transition function

(define (ext-mc s e x)

(set! (state-out-port s) '()) ;default , no port to be sent to

(set! (state-job s) (content-value x))

(case (content-port x)

('in

(cond

( (equal? (state-spfour1-s s) 'passive)

(set! (state-out-port s) 'x1)

(set! (state-spfour1-s s) 'busy))

( (equal? (state-spfour2-s s) 'passive)

(set! (state-out-port s) 'x2)

(set! (state-spfour2-s s) 'busy))

( (equal? (state-spfour3-s s) 'passive)

(set! (state-out-port s) 'x3)

(set! (state-spfour3-s s) 'busy))

))

('y1 (set! (state-spfour1-s s) 'passive)

(set! (state-out-port s) 'out)

)

('y2 (set! (state-spfour2-s s) 'passive)

(set! (state-out-port s) 'out)

)

('y3 (set! (state-spfour3-s s) 'passive)

(set! (state-out-port s) 'out)

)

)

(hold-in 'busy 0)

)

(define (int-mc s)

(case (state-phase s)

('busy

(passivate)

)))

(define (out-mc s)

(case (state-phase s)

('busy

(case (state-out-port s)

((x1 x2 x3 out)

(make-content 'port (state-out-port s)

'value (state-job s))

)

(else (make-content))

))))

(send mul-c set-ext-transfn ext-mc)

(send mul-c set-int-transfn int-mc)

(send mul-c set-outputfn out-mc)

1. dc-c

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The content of dc-c.m ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;; Divide and Conquer Co-ordinator ;;;;;;;;;;;;;;;;;;;;;;;;;

;-------------------------------------------------------------

; This file contains the definition of the co-ordinator in

; divide and conquer architecture.

;-------------------------------------------------------------

; It should perform following tasks:

; 1) Gets a problem from input and sends the problem to problem-partitioner

; 2) When the divided problem is sent back, decides whether the

; sub-processors are ALL available. If they are, the sub-problems

; will be send to all sub-processors. If not, problem is lost.

; 3) After collecting all the returned results from

; sub-processors, sends the returned partial results to

; post-compiler.

; 4) Gets the final result from compiler and sends

; it to output.

;-------------------------------------------------------------

;; make a pair for the co-ordinator in divide and conquer module

(make-pair atomic-models 'dc-c)

(send dc-c def-state '(

p-cnt ;; number of partial solutions received

out-port ;; destination for next output

job-id

first

second

third

pl\_list

))

;; initialize the states of this module

(send dc-c set-s (make-state 'sigma 'inf

'phase 'passive

'p-cnt 3

'out-port '()

'job-id '()

'first '()

'second '()

'third '()

'pl\_list '()

) )

;;;;;;;; Definition of divide and conquer co-ordinator

;; external transition function

(define (ex-dc s e x)

(set! (state-out-port s) '())

(set! (state-job-id s) (car (content-value x ) ))

(case (content-port x)

; case 1. arrival of a problem

('in

; Always send to partition processor

(set! (state-out-port s) 'px)

(set! (state-pl\_list s) (cadr (content-value x) ))

)

;case 2. input from partition processor

('py (set! (state-out-port s) 'xin)

(set! (state-first s) (cadr (content-value x) ) )

(set! (state-second s) (caddr (content-value x) ) )

(set! (state-third s) (cadddr (content-value x) ) )

)

;case 3. input from partial solution processors

('y1 (set! (state-p-cnt s) (1+ (state-p-cnt s)))

(set! (state-first s) (cdr (content-value x) ) )

(when (= (state-p-cnt s) 3)

; send the partial results to compiler

(set! (state-out-port s) 'cx)

) )

('y2 (set! (state-p-cnt s) (1+ (state-p-cnt s)))

(set! (state-second s) (cdr (content-value x) ) )

(when (= (state-p-cnt s) 3)

; send the partial results to compiler

(set! (state-out-port s) 'cx)

) )

('y3 (set! (state-p-cnt s) (1+ (state-p-cnt s)))

(set! (state-third s) (cdr (content-value x) ) )

(when (= (state-p-cnt s) 3)

; send the partial results to compiler

(set! (state-out-port s) 'cx)

) )

;case 4. input from the post compiler

('cy (set! (state-out-port s) 'out))

(set! (state-pl\_list s) (cdr (content-value x)))

) ; end of case

(hold-in 'busy 0)

) ; end of ext-transition function definition

;;;;;;;; output function

(define (out-dc s)

(case (state-phase s)

('busy

(case (state-out-port s)

('xin ;;; check whether the processors are all free

(if (= (state-p-cnt s) 3)

(begin

(set! (state-p-cnt s) 0)

;; send to three processors at the same time

(list

(make-content 'port 'x1 'value (append (list(state-job-id s) ) (list(state-first s)) ))

(make-content 'port 'x2 'value (append (list(state-job-id s) ) (list(state-second s)) ))

(make-content 'port 'x3 'value (append (list(state-job-id s) ) (list(state-third s) )))

)

); else send no partial jobs to processors

(make-content)

)

)

('px (make-content 'port (state-out-port s) 'value (append (list(state-job-id s) ) (list(state-pl\_list s) ))) )

('cx

(make-content 'port (state-out-port s) 'value (append (append (append (list(state-job-id s)) (list(state-first s)) ) (list(state-second s))) (list(state-third s))) )

)

('out

(make-content 'port 'out 'value

(append (list(state-job-id s) ) (list(state-pl\_list s))) )

)

(else (make-content)) ; no valid output to be made

) ) ) )

;;;;;;;; internal transition function

(define (in-dc s)

(case (state-phase s)

('busy

(passivate)

) ) )

;;;;;;;; assignment to the model

(send dc-c set-ext-transfn ex-dc)

(send dc-c set-int-transfn in-dc)

(send dc-c set-outputfn out-dc)

1. mul-arch.m

;;;;;;;;;;;;mul-arch.m;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The Multi-Server Architecture ;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;-------------------------------------------------------------

; This file contains the construction of the multi-server

; architecture using a digraph-model. The components

; are retrieved from prototypes in the model-base.

; Components: Three sub-processors are copies of p in file "p.m"

; One co-ordinator : mul-c defined in "mul-c.m"

;-------------------------------------------------------------

(load-from model-base\_directory pl.m)

(load-from model-base\_directory mul-c.m)

;; make three copies from original p processor and copy its initial state

(send pl make-new 'p1)

(send p1 copy-state pl)

(send pl make-new 'p2)

(send p2 copy-state pl)

(send pl make-new 'p3)

(send p3 copy-state pl)

;;now couple them to the multi-server

(make-pair digraph-models 'mul-arch)

;;composition tree

(send mul-arch build-composition-tree

mul-arch

(list mul-c p1 p2 p3))

;;influence digraph

(send mul-arch set-inf-dig (list (list mul-c p1 p2 p3)

(list p1 mul-c)

(list p2 mul-c)

(list p3 mul-c)))

;;internal coupling between processors and co-ordinator

(send mul-arch set-int-coup mul-c p1 (list (cons 'x1 'in)))

(send mul-arch set-int-coup p1 mul-c (list (cons 'out 'y1)))

(send mul-arch set-int-coup mul-c p2 (list (cons 'x2 'in)))

(send mul-arch set-int-coup p2 mul-c (list (cons 'out 'y2)))

(send mul-arch set-int-coup mul-c p3 (list (cons 'x3 'in)))

(send mul-arch set-int-coup p3 mul-c (list (cons 'out 'y3)))

;; external-input coupling

(send mul-arch set-ext-inp-coup mul-c (list (cons 'in 'in)))

;; external -output coupling

(send mul-arch set-ext-out-coup mul-c (list (cons 'out 'out)))

;; define the select function to avoid collision when a job

;; arrives at the time a processor finishes: processors first

;; then co-ordinator

(define (sel-mul slst)

(cond ((member p1 slst) p1)

((member p2 slst) p2)

((member p3 slst) p3)

((member mul-c slst) mul-c)

))

(send mul-arch set-selectfn sel-mul)

;; equivalently

(send mul-arch set-priority (list p1 p2 p3 mul-c))

;; is shorter and preferable when using flat-devs and deep-devs

1. dc-arch.m

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;Contents of file: dc-arch.m ;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The Module of Divide and Conquer Architecture

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;-------------------------------------------------------------

; This file contains the construction of the divide and conquer

; architecture by using digraph-model. The components

; are retrieved from the models.m defined in model-base.

;; components: One job partition process (p.m) -- p&div.

; Three sub-processors (p.m) -- p1, p2, p3.

; One post-compiler (p.m) -- p&cmpl.

; One co-ordinator (dc-c.m) -- dc-c.

;-------------------------------------------------------------

;; get one base processor and one co-ordinator

(load-from model-base\_directory psm.m)

(load-from model-base\_directory pc.m)

(load-from model-base\_directory pd.m)

(load-from model-base\_directory dc-c.m)

;; make five copies from the original p processor

; first the pre-job-partition processor

(send pd make-new 'p&div)

; and the post-compiler

(send pc make-new 'p&cmpl)

; three sub-processors

(send psm make-new 'p1)

(send psm make-new 'p2)

(send psm make-new 'p3)

;; now couple them together

(make-pair digraph-models 'dc-arch)

;; composition components

(send dc-arch build-composition-tree

dc-arch

(list dc-c p&div p1 p2 p3 p&cmpl)

)

;;p&div: partition processor

;;p&cmpl: compiler

;; influence digraph

(send dc-arch set-inf-dig (list (list dc-c p&div p1 p2 p3 p&cmpl)

(list p&div dc-c)

(list p1 dc-c)

(list p2 dc-c)

(list p3 dc-c)

(list p&cmpl dc-c)))

;; internal coupling

(send dc-arch set-int-coup dc-c p&div (list (cons 'px 'in)))

(send dc-arch set-int-coup p&div dc-c (list (cons 'out 'py)))

(send dc-arch set-int-coup dc-c p1 (list (cons 'x1 'in)))

(send dc-arch set-int-coup p1 dc-c (list (cons 'out 'y1)))

(send dc-arch set-int-coup dc-c p2 (list (cons 'x2 'in)))

(send dc-arch set-int-coup p2 dc-c (list (cons 'out 'y2)))

(send dc-arch set-int-coup dc-c p3 (list (cons 'x3 'in)))

(send dc-arch set-int-coup p3 dc-c (list (cons 'out 'y3)))

(send dc-arch set-int-coup dc-c p&cmpl (list (cons 'cx 'in)))

(send dc-arch set-int-coup p&cmpl dc-c (list (cons 'out 'cy)))

;; external-internal coupling

(send dc-arch set-ext-inp-coup dc-c (list (cons 'in 'in)))

;; internal-external coupling

(send dc-arch set-ext-out-coup dc-c (list (cons 'out 'out)))

;; define the select function to avoid collision when a job

;; arrives at the same time the processor finishes

(define (sel-dcc slst)

(cond ((member p&cmpl slst) p&cmpl)

((member p1 slst) p1)

((member p2 slst) p2)

((member p3 slst) p3)

((member p&div slst) p&div)

((member dc-c slst) dc-c)

)

)

(send dc-arch set-selectfn sel-dcc)

;;equivalently, and preferably when flat-devs and deep-devs are used

(send dc-arch set-priority (list p&cmpl p1 p2 p3 p&div dc-c))

1. ef.m

;;;;;;;; ef.m ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The module of a global experimental frame ;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;-------------------------------------------------------------

; This file contains the construction of the experiment frame

; by using digraph-models. The components

; are retrieved from the models.m defined in model-base.

;; components: One generator (genr.m) -- genr

; one transducer (transd.m) -- transd

;-------------------------------------------------------------

;; Two component of experimental frame, generator and transducer

(load-from model-base\_directory genr.m)

(load-from model-base\_directory transd.m)

;; couple them in a digraph-model

(make-pair digraph-models 'ef)

(send ef build-composition-tree ef (list genr transd))

(send ef set-inf-dig (list (list genr transd) (list transd genr)))

;; the connection from transducer to generator is made

;; for the termination of experiment

(send ef set-int-coup transd genr (list (cons 'out 'stop)))

(send ef set-int-coup genr transd (list (cons 'out 'ariv)))

(send ef set-ext-inp-coup transd (list (cons 'in 'solved)))

(send ef set-ext-out-coup genr (list (cons 'out 'out)))

(send ef set-ext-out-coup transd (list (cons 'out 'result)))

1. ef-pl.m

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Model/Frame pair ef-p.m ;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;-------------------------------------------------------------

; This file uses a digraph model to couple the simple processor

; with the experimental frame.

;-------------------------------------------------------------

;; load the components

(load-from model-base\_directory pl.m)

(load-from model-base\_directory ef.m)

;; couple the experimental frame with the processor by p-ef

(make-pair digraph-models 'ef-pl)

;; the composition tree

(send ef-pl build-composition-tree ef-pl (list pl ef))

;; the influence digraph

(send ef-pl set-inf-dig (list (list pl ef)

(list ef pl)

) )

;; internal coupling

(send ef-pl set-int-coup pl ef (list (cons 'out 'in)))

(send ef-pl set-int-coup ef pl (list (cons 'out 'in)))

(send ef-pl set-ext-out-coup ef (list (cons 'result 'out)))

;; define the select function to avoid collision when the job

;; arrives at the time the processor finishes: processor first

;; then generator

(define (sel-pl slst)

(cond ((member pl slst) pl)

((member ef slst) ef)

) )

(send ef-pl set-selectfn sel-pl)

;; equivalently

(send ef-pl set-priority (list pl ef))

;; is shorter and preferable when using flat-devs and deep-devs

;; the final touch, attach a root co-ordinator

(mk-ent root-co-ordinators r)

;; initialize it with the co-ordinator for ef-p

(initialize r c:ef-pl)

;; start a simulation run

(restart r)

1. ef-mul.m

(load-from model-base\_directory mul-arch.m)

(load-from model-base\_directory ef.m)

(make-pair digraph-models 'ef-mul)

(send ef-mul build-composition-tree ef-mul (list mul-arch ef))

(send ef-mul set-inf-dig(list (list mul-arch ef) (list ef mul-arch) ) )

(send ef-mul set-int-coup mul-arch ef (list (cons 'out 'in) ) )

(send ef-mul set-int-coup ef mul-arch (list (cons 'out 'in) ) )

(send ef-mul set-ext-out-coup ef (list (cons 'result 'out)))

(define (sel-mul slst)

(cond ( (member mul-arch slst) mul-arch)

( ( member ef slst) ef)

) )

(send ef-mul set-selectfn sel-mul)

(send ef-mul set-priority (list mul-arch ef) )

(mk-ent root-co-ordinators r)

(initialize r c:ef-mul)

(restart r)

1. ef-dc.m

(load-from model-base\_directory DC-ARCH.m)

(load-from model-base\_directory ef.m)

(make-pair digraph-models 'ef-dc)

(send ef-dc build-composition-tree ef-dc (list dc-arch ef))

(send ef-dc set-inf-dig(list (list dc-arch ef) (list ef dc-arch) ) )

(send ef-dc set-int-coup dc-arch ef (list (cons 'out 'in) ) )

(send ef-dc set-int-coup ef dc-arch (list (cons 'out 'in) ) )

(send ef-dc set-ext-out-coup ef (list (cons 'result 'out)))

(define (sel-dc slst)

(cond ( (member dc-arch slst) dc-arch)

( ( member ef slst) ef)

) )

(send ef-dc set-selectfn sel-dc)

(send ef-dc set-priority (list ef-dc ef) )

(mk-ent root-co-ordinators r)

(initialize r c:ef-dc)

(restart r)

SES source codes

* entity을 추가할 때 p 대신 pl 을 적으면 pl\_arch이 없는 문제가 생겨 entity추가는 그대로 p로 하고 p의 세부내용을 mbase에 있는 p.m의 세부 내용을 pl의 내용과 같게 했습니다.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; ef-a.s ENTITY STRUCTURE FOR SIMPLE ARCHITECTURES ;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;-------------------------------------------------------------

;; this file contains the structure construction of the experimental

;; frame and all four architectures.

;;;-------------------------------------------------------------

;; create the root node -- entstr of ef ef-a

(make-entstr 'ef-a)

;; add a level of decomposition

(add-item e:ef-a asp 'ef-a-dec)

;; set cursor to ef-a-dec

;; and describe ef-a-dec

(set-current-item e:ef-a 'ef-a-dec)

;; add its components -- arch and ef

(add-item e:ef-a ent 'arch)

(add-item e:ef-a ent 'ef)

;-------- coupling -----------------------------------

(add-couple e:ef-a 'arch 'ef 'out 'in)

(add-couple e:ef-a 'ef 'arch 'out 'in)

(add-couple e:ef-a 'ef 'ef-a 'result 'out)

;-----------------------------------------------------

;; priority based selectfn

(add-priority e:ef-a '(arch ef))

;; now describing arch

(set-current-item e:ef-a 'arch)

;; arch can be any of the architecture under test

(add-item e:ef-a spec 'arch-spec)

(set-current-item e:ef-a 'arch-spec)

;; and those architectures under test are --

(add-item e:ef-a ent 'p)

(add-item e:ef-a ent 'mul)

(add-item e:ef-a ent 'd&c)

;;; the descriptions of pipeline architecture

;;; -- pip

;;; the descriptions of multi-server architecture

;;; -- mul

(set-current-item e:ef-a 'mul)

(add-item e:ef-a asp 'mul-dec)

(set-current-item e:ef-a 'mul-dec)

(add-item e:ef-a ent 'mul-c)

(add-item e:ef-a ent 'p1)

(add-item e:ef-a ent 'p2)

(add-item e:ef-a ent 'p3)

;-------- coupling -----------------------------------

(add-couple e:ef-a 'mul 'mul-c 'in 'in)

(add-couple e:ef-a 'mul-c 'mul 'out 'out)

(add-couple e:ef-a 'mul-c 'p1 'x1 'in)

(add-couple e:ef-a 'mul-c 'p2 'x2 'in)

(add-couple e:ef-a 'mul-c 'p3 'x3 'in)

(add-couple e:ef-a 'p1 'mul-c 'out 'y1)

(add-couple e:ef-a 'p2 'mul-c 'out 'y2)

(add-couple e:ef-a 'p3 'mul-c 'out 'y3)

;; priority based selectfn

(add-priority e:ef-a '(p1 p2 p3 mul-c))

;;; the descriptions of divide and conquer architecture

;;; -- d&c ;;can't use dc since (dc) exits

(set-current-item e:ef-a 'd&c)

(add-item e:ef-a asp 'd&c-dec)

(set-current-item e:ef-a 'd&c-dec)

(add-item e:ef-a ent 'dc-c)

(add-item e:ef-a ent 'p1)

(add-item e:ef-a ent 'p2)

(add-item e:ef-a ent 'p3)

(add-item e:ef-a ent 'p&div)

(add-item e:ef-a ent 'p&cmpl)

;-------- coupling -----------------------------------

(add-couple e:ef-a 'd&c 'dc-c 'in 'in)

(add-couple e:ef-a 'dc-c 'd&c 'out 'out)

(add-couple e:ef-a 'dc-c 'p1 'x1 'in)

(add-couple e:ef-a 'dc-c 'p2 'x2 'in)

(add-couple e:ef-a 'dc-c 'p3 'x3 'in)

(add-couple e:ef-a 'dc-c 'p&div 'px 'in)

(add-couple e:ef-a 'dc-c 'p&cmpl 'cx 'in)

(add-couple e:ef-a 'p1 'dc-c 'out 'y1)

(add-couple e:ef-a 'p2 'dc-c 'out 'y2)

(add-couple e:ef-a 'p3 'dc-c 'out 'y3)

(add-couple e:ef-a 'p&div 'dc-c 'out 'py)

(add-couple e:ef-a 'p&cmpl 'dc-c 'out 'cy)

;; priority based selectfn

(add-priority e:ef-a '(p&cmpl p3 p2 p1 p&div dc-c))

;; describe the experimental frame structure

(set-current-item e:ef-a 'ef)

(add-item e:ef-a asp 'ef-dec)

;; experimental frame consists of generator and transducer

(set-current-item e:ef-a 'ef-dec)

(add-item e:ef-a ent 'transd)

(add-item e:ef-a ent 'genr)

;-------- coupling -----------------------------------

(add-couple e:ef-a 'ef 'transd 'in 'solved)

(add-couple e:ef-a 'transd 'ef 'out 'result)

(add-couple e:ef-a 'transd 'genr 'out 'stop)

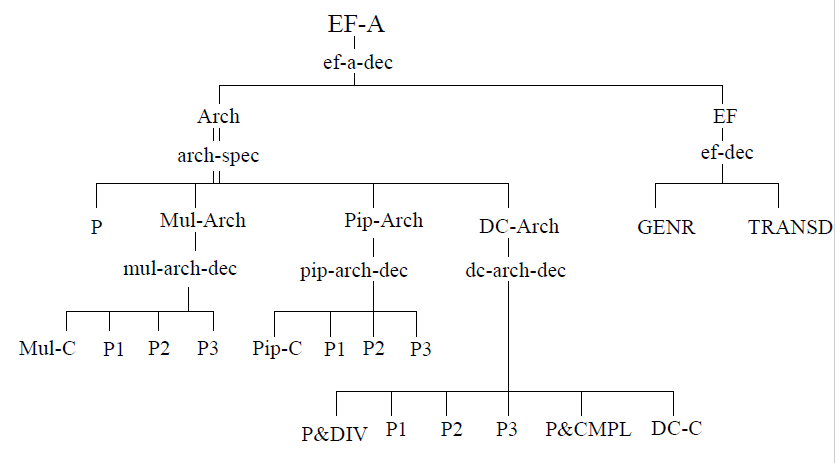
(add-couple e:ef-a 'genr 'ef 'out 'out)

(add-couple e:ef-a 'genr 'transd 'out 'ariv)

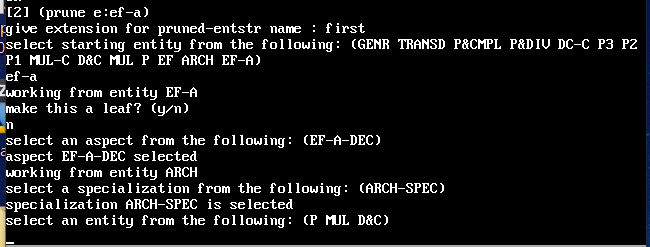
;; save the entity structure

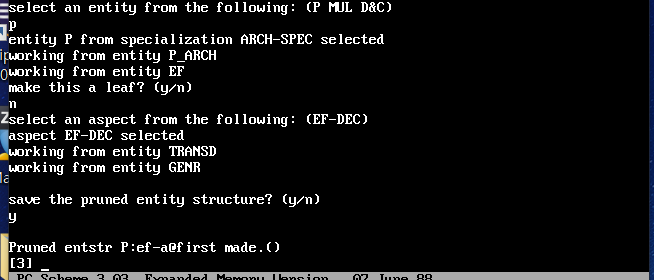
(save-en e:ef-a)

SES diagram



Prune





Transform



