Realizzazioni software di modelli dataflow

Lezione 04 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2016-17

Indice

argomenti della lezione

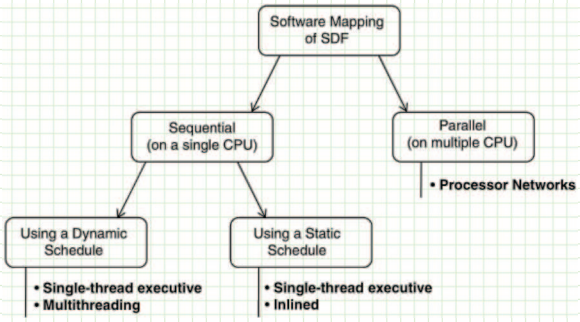
di che si tratta:

- panoramica di metodi di realizzazione software di modelli dataflow
- realizzazione di code FIFO in C
- realizzazione di attori in C
- esempio: modello SDF di una FFT e sua realizzazione
- scheduling dinamico con multithreading cooperativo
- uso della libreria QuickThreads
- ottimizzazione della realizzazione con scheduling statico e inlining

metodi di realizzazione software di modelli dataflow

elementi del modello da tradurre in software: attori, code, regole di attivazione

ampio spettro di opzioni di realizzazione:



Schaumont, Figure 3.1 - Overview of possible approaches to map dataflow into software

realizzazioni parallele, ottimizzazione dell'assegnamento di attori ai processori:

- bilanciamento del carico computazionale
- minimizzazione della comunicazione fra processori

realizzazioni sequenziali, opzioni: scheduling, threading

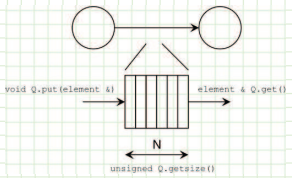
scheduling statico → ulteriori opportunità di ottimizzazione

realizzazione delle code FIFO

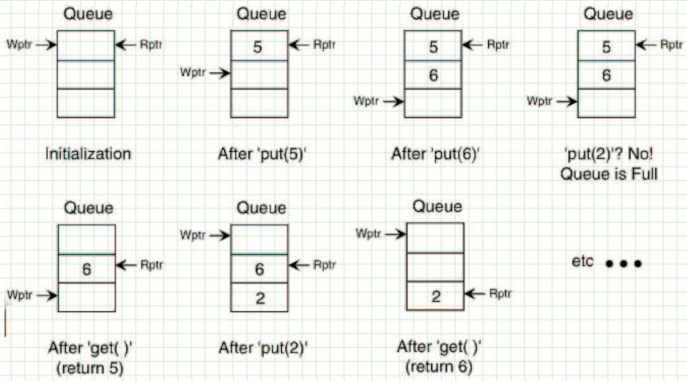
coda FIFO, struttura con:

- due parametri: capacità, tipo di dato degli elementi
- tre operazioni: void put(element), element get(), int getsize()

si può realizzare mediante un array circolare e due puntatori alle locazioni di accesso per le operazioni di lettura e scrittura, incrementati mod N se la dimensione dell'array è N+1



Schaumont, Figure 3.2 - A software queue



Schaumont, Figure 3.3 - Operation of the circular queue

coda FIFO in C

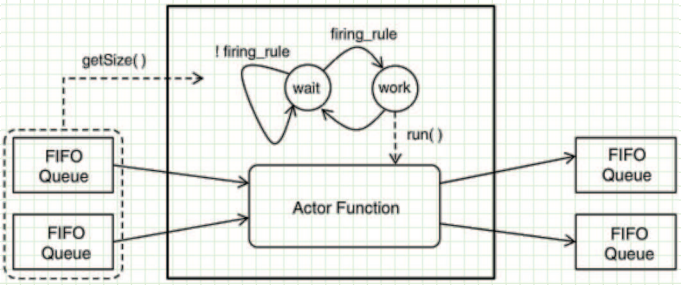
```
#define MAXFIFO 8
typedef struct fifo {
  int data[MAXFIFO]; // token storage
  unsigned wptr; // write pointer
  unsigned rptr; // read pointer
} fifo_t;

void init_fifo(fifo_t *F) {
  F->wptr = F->rptr = 0;
}

void put_fifo(fifo_t *F, int d) {
  if (((F->wptr + 1) % MAXFIFO) != F->rptr) {
    F->data[F->wptr] = d;
    F->wptr = (F->wptr + 1) % MAXFIFO;
  }
}

int get_fifo(fifo_t *F) {
  int r;
  if (F->rptr != F->wptr) {
    r = F->data[F->rptr];
    F->rptr = (F->rptr + 1) % MAXFIFO;
    return r;
  }
  return -1;
}

unsigned fifo_size(fifo_t *F) {
  if (F->wptr >= F->rptr)
    return F->wptr - F->rptr;
  else
    return MAXFIFO - (F->rptr - F->wptr);
}

int main() {
  fifo_t F1;
  init_fifo(&F1); // resets wptr, rptr;
  put_fifo(&F1, 5); // enter 5
  put_fifo(&F1, 6); // enter 6
  printf("%d %d\n", fifo_size(&F1), get_fifo(&F1));
  // prints: 2 5
  printf("%d\n", fifo_size(&F1)); // prints: 1
}
```

Schaumont, Listing 3.1 - FIFO object in C

realizzazione degli attori

una funzione C, parametrizzata con una struttura dati di supporto all'I/O sulle code FIFO

- l'I/O dell'attore è condizionato dalla validità della sua regola di attivazione
  presenza del numero richiesto di token in ciascuna delle sue code d'ingresso
- la validità delle regole di attivazione è controllata dall'attore a ogni sua invocazione
  un primo esempio elementare di FSM con datapath (FSMD), v. figura



Schaumont, Figure 3.4 - Software implementation of the datafow actor

esempio di attore in C

una struttura dati di supporto per max. otto code d'ingresso e altrettante di uscita:

```
#define MAXIO 8
typedef struct actorio {
  fifo_t *in[MAXIO];
  fifo_t *out[MAXIO];
} actorio_t;
```

un attore che accetta due token interi e ne produce la somma e la differenza:

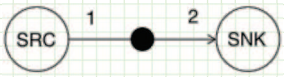
```
void fft2(actorio_t *g) {
  int a, b;
  if (fifo_size(g->in[0]) >= 2) {
    a = get_fifo(g->in[0]);
    b = get_fifo(g->in[0]);
    put_fifo(g->out[0], a+b);
    put_fifo(g->out[0], a-b);
  }
}
```

con questo schema, gli attori possono essere istanziati nel programma e invocati con scheduling dinamico

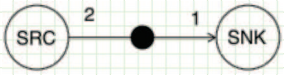
realizzazione software con scheduler dinamico

uno scheduler dinamico di sistema istanzia e inizializza attori e code, quindi invoca gli attori, per esempio a turno uno per volta:

```
void main() {
 fifo_t q1, q2;
 actorio_t fft2_io = {{&q1}, {&q2}};
 ..
 init_fifo(&q1);
 init_fifo(&q2);
 ..
 while (1) {
  fft2_actor(&fft2_io);
  // .. call other actors
 }
}
```



Schaumont, Figure 3.5a - A graph which will simulate under a single rate system schedule



Schaumont, Figure 3.5b - A graph which will cause extra tokens under a single rate schedule

schedule di sistema

```
void main() {
 ..
 while (1) {
  src_actor(&src_io);
  snk_actor(&snk_io);
 }
}
```

è evidente un problema con l'esempio nella seconda figura...

soluzione 1: adattare la schedule di sistema

```
void main() {
 ..
 while (1) {
  src_actor(&src_io);
  snk_actor(&snk_io);
  snk_actor(&snk_io);
 }
}
```

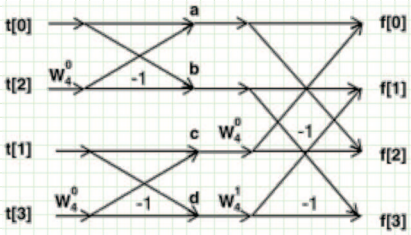
soluzione 2: adattare il codice dell'attore `snk`

```
void snk_actor(actorio_t *g) {
 int r1, r2;
 while ((fifo_size(g->in[0]) > 0)) {
  r1 = get_fifo(g->in[0]);
  ... // do processing
 }
}
```


esempio: modello SDF di una FFT

la trasformata rapida (discreta) di Fourier è largamente usata nell'elaborazione di segnali

- *fattori twiddle* (radici complesse dell'unità): $W_N^k = W(k, N) = e^{-j2\pi k/N}$



Schaumont, Figure 3.6a - Flow diagram for a four-point Fast Fourier Transform

a = t[0] + W(0,4) * t[2] = t[0] + t[2]
b = t[0] - W(0,4) * t[2] = t[0] - t[2]
c = t[1] + W(0,4) * t[3] = t[1] + t[3]
d = t[1] - W(0,4) * t[3] = t[1] - t[3]
f[0] = a + W(0,4) * c = a + c
f[1] = b + W(1,4) * d = b - j.d
f[2] = a - W(0,4) * c = a - c
f[3] = b - W(1,4) * d = b + j.d

un modello SDF per il calcolo delle ampiezze nel dominio della frequenza:

Schaumont, Figure 3.7 - Synchronous dataflow diagram for a four-point Fast Fourier Transform

- reorder : (t[0],t[1],t[2],t[3]) → (t[0],t[2],t[1],t[3])
- fft2 : l'attore in esempio a p. 7
- fft4mag : → ampiezze di 4 punti della trasformata

realizzazione software del modello di FFT

```
void reorder(actorio_t *g) {
  int v0, v1, v2, v3;
  while (fifo_size(g->in[0]) >= 4) {
    v0 = get_fifo(g->in[0]);
    v1 = get_fifo(g->in[0]);
    v2 = get_fifo(g->in[0]);
    v3 = get_fifo(g->in[0]);
    put_fifo(g->out[0], v0);
    put_fifo(g->out[0], v2);
    put_fifo(g->out[0], v1);
    put_fifo(g->out[0], v3);
  }
}
```

```
void fft2(actorio_t *g) {
  int a, b;
  while (fifo_size(g->in[0]) >= 2) {
    a = get_fifo(g->in[0]);
    b = get_fifo(g->in[0]);
    put_fifo(g->out[0], a+b);
    put_fifo(g->out[0], a-b);
  }
}
```

```
void fft4mag(actorio_t *g) {
  int a, b, c, d;
  while (fifo_size(g->in[0]) >= 4) {
    a = get_fifo(g->in[0]);
    b = get_fifo(g->in[0]);
    c = get_fifo(g->in[0]);
    d = get_fifo(g->in[0]);
    put_fifo(g->out[0], (a+c)*(a+c));
    put_fifo(g->out[0], b*b - d*d);
    put_fifo(g->out[0], (a-c)*(a-c));
    put_fifo(g->out[0], b*b - d*d);
  }
}
```

```
int main() {
  fifo_t q1, q2, q3, q4;
  actorio_t reorder_io = {{&q1}, {&q2}};
  actorio_t fft2_io = {{&q2}, {&q3}};
  actorio_t fft4_io = {{&q3}, {&q4}};

  init_fifo(&q1);
  init_fifo(&q2);
  init_fifo(&q3);
  init_fifo(&q4);

  // test vector fft([1 1 1 1])
  put_fifo(&q1, 1);
  put_fifo(&q1, 1);
  put_fifo(&q1, 1);
  put_fifo(&q1, 1);
  // test vector fft([1 1 1 0])
  put_fifo(&q1, 1);
  put_fifo(&q1, 1);
  put_fifo(&q1, 1);
  put_fifo(&q1, 0);

  while (1) {
    reorder(&reorder_io);
    fft2 (&fft2_io);
    fft4mag(&fft4_io);
  }

  return 0;
}
```

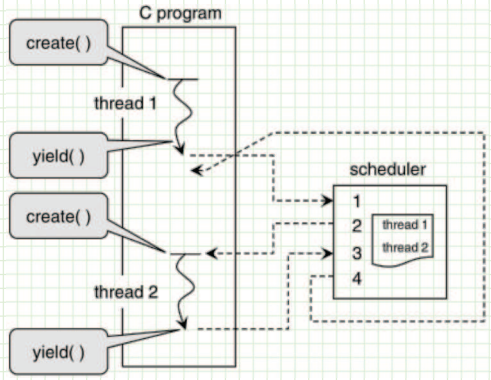
Schaumont, Listing 3.2 - 4-point FFT as an SDF system

scheduling dinamico con multithreading cooperativo

con il multithreading si realizza lo scheduling dinamico assegnando a ciascun attore un proprio thread di esecuzione

- multithreading *cooperativo*: rilascio del controllo e riattivazione dell'esecuzione dal punto in cui l'attore ha effettuato il rilascio

spesso: scheduler round-robin + soluzione 2 vista prima



funzioni di libreria QuickThreads:

- stp_init()
- stp_create(stp_userf_t *F, void *G)
- stp_start()
- stp_yield()
- stp_abort()

Schaumont, Figure 3.8 - Example of cooperative multi-threading

realizzazione con uso della libreria QuickThreads

il rilascio del controllo con `stp_yield()` mantiene le variabili locali attraverso esecuzioni successive

```
#include "../qt/stp.h"
#include <stdio.h>
void hello(void *null) {
  int n = 3;
  while (n-- > 0) {
    printf("hello\n");
    stp_yield();
  }
}
void world(void *null) {
  int n = 5;
  while (n-- > 0) {
    printf("world\n");
    stp_yield();
  }
}
int main(int argc, char **argv) {
  stp_init();
  stp_create(hello, 0);
  stp_create(world, 0);
  stp_start();
  return 0;
}
```

l'applicazione di questa tecnica all'esempio sviluppato nel Listing 3.2 è molto semplice, e.g. per l'attore `fft2`:

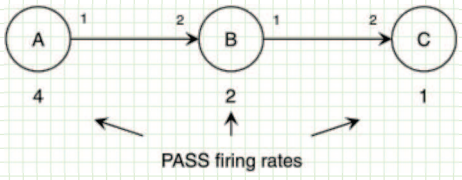
```
void fft2(actorio_t *g) {
  int a, b;
  while (1) {
    while (fifo_size(g->in[0]) >= 2) {
      a = get_fifo(g->in[0]);
      b = get_fifo(g->in[0]);
      put_fifo(g->out[0], a+b);
      put_fifo(g->out[0], a-b);
    }
    stp_yield();
  }
}

int main() {
  fifo_t q1, q2;
  actorio_t fft2_io = {{&q1}, {&q2}};
  ...
  stp_create(fft2, &fft2_io); // create thread
  ...
  stp_start(); // run the schedule
}
```


ottimizzazione della realizzazione con scheduling statico

lo scheduling statico permette di ottimizzare una realizzazione software in tre aspetti:

- eliminazione del controllo delle condizioni di attivazione negli attori
- ottimizzazione della schedule minimizzando la capacità delle code FIFO
- *inlining* del software:
  - rimpiazzamento delle code con variabili → risparmio di spazio
  - eliminazione delle chiamate di funzione → risparmio di tempo

esempio:




```
while(1) {
  // call A four times
  A(); A(); A(); A();
  // call B two times
  B(); B();
  // call C one time
  C();
}
```

Schaumont, Figure 3.9 - System schedule for a multirate SDF graph

la schedule statica in figura è una PASS, tuttavia non è ottimale per la minimizzazione delle capacità delle code: lo è la sequenza di attivazione (A, A, B, A, A, B, C)

realizzazione ottimizzata del modello di FFT4 a p. 9 con schedule statica e inlining:

```
void dfsystem(int in0, in1, in2, in3, *out0, *out1, *out2, *out3) {
  int reorder_out0, reorder_out1, reorder_out2, reorder_out3;
  int fft2_0_out0, fft2_0_out1, fft2_1_out0, fft2_1_out1;
  int fft4mag_out0, fft4mag_out1, fft4mag_out2, fft4mag_out3;

  reorder_out0 = in0;
  reorder_out1 = in2;
  reorder_out2 = in1;
  reorder_out3 = in3;

  fft2_0_out0 = reorder_out0 + reorder_out1;
  fft2_0_out1 = reorder_out0 - reorder_out1;
  fft2_1_out0 = reorder_out2 + reorder_out3;
  fft2_1_out1 = reorder_out2 - reorder_out3;

  out0 = fft4mag_out0 = (fft2_0_out0 + fft2_1_out0) * (fft2_0_out0 + fft2_1_out0);
  out1 = fft4mag_out1 = fft2_0_out1*fft2_0_out1 - fft2_1_out1*fft2_1_out1;
  out2 = ftt4mag_out2 = (fft2_0_out0 - fft2_1_out0) * (fft2_0_out0 - fft2_1_out0);
  out3 = fft4mag_out3 = fft2_0_out1*fft2_0_out1 - fft2_1_out1*fft2_1_out1;
}
```

Schaumont, Listing 3.4 – Inlined data flow system for the four-point FFT

letture raccomandate:

Schaumont (2012) Cap. 3, Sez. 3.1

per ulteriore consultazione:

Schaumont (2012) Cap. 4