FLUENT-UDF 高级应用技术培训－实例

2005 年 6 月 9 日～10 日
飞昂软件技术（上海）有限公司

例一

```
;;
;; FLUENT User Defined Scheme Program
;;
;; Define User Scheme Variables
;;
;; Fluent Inc.

(define (set-new-var s v t)
   (if (not (rp-var-object s))
         (rp-var-define s v t #f)
         )
   )

(for-each
  (lambda (var) (apply set-new-var var))
  '(
     (my/int        1           integer) ;user integer
     (my/real      0.5        real)        ;user real
     (my/text      "abc"    string)    ;user text
     )
  )

;;
;; Create a panel for the udf
;;
(define gui-my-input-panel
   (let ((panel #f)
            ;declare variables - assign values later
      (mybox)
      (my-int-entry)
      (my-real-entry)
      (my-text-entry)
      )

;
; Function which is called when panel is opened
; sets panel variables to values in set-new-var section
;
      (define (update-cb . args)
         (cx-set-integer-entry    my-int-entry    (rpgetvar 'my/int))
         (cx-set-real-entry         my-real-entry (rpgetvar 'my/real))
         (cx-set-text-entry         my-text-entry (rpgetvar 'my/text))
         )

;;;
;;; Function which is called when OK button is hit.
;;; assigns variable values that were set in panel after clicking "ok"
;;;
        (define (apply-cb . args)
            (rpsetvar 'my/int    (cx-show-integer-entry my-int-entry))
            (rpsetvar 'my/real (cx-show-real-entry        my-real-entry))
            (rpsetvar 'my/text (cx-show-text-entry        my-text-entry))
            )

        (lambda args
;;; if panel does not exist, make panel
            (if (not panel)
            ;; create panel

            (let ((table))
                (set! panel (cx-create-panel "My Input Panel" apply-cb update-cb))
                ;; create table w/o title in panel to enable row by column format
                (set! table (cx-create-table panel "" 'border #f 'below 0 'right-of 0))

;;; Create Custom Variable input.
                (set! mybox (cx-create-table table "My Inputs" 'border #t 'row 0))
                (set! my-int-entry    (cx-create-integer-entry mybox "my integer" 'row 0 'col 0))
                (set! my-real-entry (cx-create-real-entry        mybox "my real"        'row 1 'col 0))
                (set! my-text-entry (cx-create-text-entry        mybox "my text"        'row 2 'col 0))
                )
            )
            ;; tell solver to display the panel
            (cx-show-panel panel)
            )
        )
    )

;;
;; Add new panel to end of User-Defined menu.
;; no need to modify here except panel name and menu location
;;
(cx-add-item
  "Define"
  "My Input Panel..."
  #\U
  #f
  cx-client?
  gui-my-input-panel
  )
```

例二

```
#include "udf.h"

int get_species1(Domain *d)
{
   Material *mix = mixture_material(d); /* Get mixture material */
   Material *sp;
   int i, n_species;
   real mw[MAX_SPE_EQNS];
   char *sp_name;

   Message("\nIndex\tName\tMW\n");
   n_species = 0; /* Reset species counter */
   /* Loop over species in mixture */
   mixture_species_loop(mix, sp, i) {
      /* Get name of species i */
      sp_name = MIXTURE_SPECIE_NAME(mix, i);
      /* Get species molecular weight */
      mw[i] = MATERIAL_PROP(sp, PROP_mwi);
      /* Display species data */
      Message("%d\t%s\t%f\n", i, sp_name, mw[i]);
      /* Update species counter */
      n_species += 1;
   }
   /* Return with total number of species */
   return n_species;
}

int get_species2(Domain *d)
{
   Material *mix = mixture_material(d); /* Get mixture material */
   int i, n_species;
   real hf[MAX_SPE_EQNS];
   char *sp_name;

   Message("\nIndex\tName\th_form\n");
   /* Get total number of species in mixture */
   n_species = MIXTURE_NSPECIES(mix);
   /* Loop over species from 0 to n_species - 1 */
   spe_loop(i, n_species) {
      Material *sp = MIXTURE_SPECIE(mix, i); /* Get species material pointer */
      char *sp_name = MIXTURE_SPECIE_NAME(mix, i); /* Get species name */

      /* Get species hentalpy of formation */
      hf[i] = MATERIAL_PROP(sp, PROP_hform);
      /* Display species data */
      Message("%d\t%s\t%f\n", i, sp_name, hf[i]);
   }
   /* Return with total number of species */
   return n_species;
}
```

```
int find_species(Domain *d, char *sp_name)
{
    Material *mix = mixture_material(d); /* Get mixture material */
    int sp_index;

    /* Get named species index */
    sp_index = mixture_specie_index(mix, sp_name);
    /* Return with species index */
    return sp_index;
}

int get_reactions(Domain *d)
{
    Material *mix = mixture_material(d); /* Get mixture material */
    Reaction *r_list = mix->reaction_list; /* Get reaction list in mixture */
    Reaction *r;
    Material *sp;
    int n_reactions, n_r, n_p, i, j, m;
    char *r_name, *sp_name;
    real mw;

    /* Loop over reactions in the reaction list */
    n_reactions = 0;
    loop(r, r_list) {
        r_name = r->name; /* Get reaction name */
        n_r = r->n_reactants; /* Get number of reactants */
        n_p = r->n_products;    /* Get number of products */

        /* Display Reaction data */
        Message("\nReaction name: %s\n", r_name);
        Message("Reactants:\n");
        for (i = 0; i < n_r; i++) {
            sp = r->mat_reactant[i]; /* Get reactant species */
            j = r->reactant[i]; /* Get species index */
            m = r->stoich_reactant[i]; /* Get stoich constant */
            sp_name = MIXTURE_SPECIE_NAME(mix, j); /* get species name */
            mw = MATERIAL_PROP(sp, PROP_mwi);
            Message("%d\t%s\t%d\t%f\n", j, sp_name, m, mw);
        }

        Message("Products:\n");
        for (i = 0; i < n_p; i++) {
            sp = r->mat_product[i]; /* Get product species */
            j = r->product[i]; /* Get species index */
            m = r->stoich_product[i]; /* Get stoich constant */
            sp_name = MIXTURE_SPECIE_NAME(mix, j); /* get species name */
            mw = MATERIAL_PROP(sp, PROP_mwi);
            Message("%d\t%s\t%d\t%f\n", j, sp_name, m, mw);
        }
        n_reactions += 1;
    }
    return n_reactions;
}
```

```
DEFINE_ON_DEMAND(list_species)
{
    Domain *fl_domain = Get_Domain(1); /* Get domain */
    int ns;
    int method = RP_Get_Integer("spe/method");

    switch (method) {
    case 1:
        ns = get_species1(fl_domain);
        break;

    case 2:
        ns = get_species2(fl_domain);
        break;

    default:
        Error("Unknown method!\n");
        break;
    }
    Message("Total number of species: %d\n", ns);
}

DEFINE_ON_DEMAND(locate_species)
{
    Domain *fl_domain = Get_Domain(1); /* Get domain */
    int sp_index;
    char *sp_name = RP_Get_String("spe/name");

    sp_index = find_species(fl_domain, sp_name);

    if (sp_index != -1)
        Message("%s has index: %d\n", sp_name, sp_index);
    else
        Message("Sorry! Didn't find species %s in mixture!\n", sp_name);
}

DEFINE_ON_DEMAND(list_reactions)
{
    Domain *fl_domain = Get_Domain(1); /* Get domain */
    int nr;

    nr = get_reactions(fl_domain);
    Message("Total number of reactions: %d\n", nr);
}
```

例三

```
#include "udf.h"
#include "sg.h"

#define FLUID_ID 1
#define ua1 -7.1357e-2
#define ua2 54.304
#define ua3 -3.1345e3
#define ua4 4.5578e4
#define ua5 -1.9664e5

#define va1 3.1131e-2
#define va2 -10.313
#define va3 9.5558e2
#define va4 -2.0051e4
#define va5 1.1856e5

#define ka1 2.2723e-2
#define ka2 6.7989
#define ka3 -424.18
#define ka4 9.4615e3
#define ka5 -7.7251e4
#define ka6 1.8410e5

#define da1 -6.5819e-2
#define da2 88.845
#define da3 -5.3731e3
#define da4 1.1643e5
#define da5 -9.1202e5
#define da6 1.9567e6

DEFINE_PROFILE(fixed_u, thread, np)
{
   cell_t c;
   real x[ND_ND];
   real r;

   begin_c_loop (c,thread)
      {
/* centroid is defined to specify position dependent profiles*/
         C_CENTROID(x,c,thread);
         r =x[1];
         F_PROFILE(c,thread,np) =
      ua1+(ua2*r)+(ua3*r*r)+(ua4*r*r*r)+(ua5*r*r*r*r);
}
   end_c_loop (c,thread)
}


DEFINE_PROFILE(fixed_v, thread, np)
{
   cell_t c;
```

```
    real x[ND_ND];
    real r;

    begin_c_loop (c,thread)
        {
/* centroid is defined to specify position dependent profiles*/
            C_CENTROID(x,c,thread);
            r =x[1];
            F_PROFILE(c,thread,np) =
        va1+(va2*r)+(va3*r*r)+(va4*r*r*r)+(va5*r*r*r*r);
}
    end_c_loop (c,thread)
}



DEFINE_PROFILE(fixed_ke, thread, np)
{
    cell_t c;
    real x[ND_ND];
    real r;

    begin_c_loop (c,thread)
        {
/* centroid is defined to specify position dependent profiles*/
            C_CENTROID(x,c,thread);
            r =x[1];
            F_PROFILE(c,thread,np) =
            ka1+(ka2*r)+(ka3*r*r)+(ka4*r*r*r)+(ka5*r*r*r*r)+(ka6*r*r*r*r*r);

        }
    end_c_loop (c,thread)
}


DEFINE_PROFILE(fixed_diss, thread, np)
{
    cell_t c;
    real x[ND_ND];
    real r;

    begin_c_loop (c,thread)
        {
/* centroid is defined to specify position dependent profiles*/
            C_CENTROID(x,c,thread);
            r =x[1];
            F_PROFILE(c,thread,np) =
            da1+(da2*r)+(da3*r*r)+(da4*r*r*r)+(da5*r*r*r*r)+(da6*r*r*r*r*r);

        }
    end_c_loop (c,thread)
}
```

```
/*
 * Example UDF for Multiphase Flows
 *
 * FLUENT UDF Seminar, Beijing 2003
 */
#include "udf.h"

enum {
   WATER = 2,
   SAND
};

/*
 * Method 1: Loop over sub-domains
 */
int set_phase_temp1(Domain *mix_d)
{
   Domain *sub_d;
   Thread *mix_t, *sub_t;
   cell_t c;
   int p_d_idx, p_d_id, d_index, n_sub_domains;
   real T;

   /* Get number of sub-domains in mixture */
   n_sub_domains = DOMAIN_N_DOMAINS(mix_d);
   /* Loop over sub-domains in mixture domain */
   sub_domain_loop(sub_d, mix_d, p_d_idx) {
      p_d_id = DOMAIN_ID(sub_d);
      d_index = PHASE_DOMAIN_INDEX(sub_d); /* Get phase domain index */
      Message("\nThis is phase domain: %d; %d\n", p_d_id, d_index);
      if (p_d_id == WATER)
         T = 300.;
      else if (p_d_id == SAND)
         T = 1000.;
      else
         T = 500.;
      /* Loop over cell thread in sub-domain */
      thread_loop_c(sub_t, sub_d) {
         if (NNULLP(THREAD_STORAGE(sub_t, SV_T))) { /* Check memory for T */
      begin_c_loop(c, sub_t) {
         C_T(c, sub_t) = T;
      } end_c_loop(c, sub_t);
         }
      }
   }
   return n_sub_domains;
}

int set_phase_temp2(Domain *mix_d)
{
   Domain *sub_d;
   Thread *mix_t, *sub_t, **phs_t;
   cell_t c;
```

```
    int p_d_idx, p_d_id, d_index, n_sub_domains;
    real T;

    /* Get number of sub-domains in mixture */
    n_sub_domains = DOMAIN_N_DOMAINS(mix_d);

    /* Loop over cell thread in mixture domain */
    mp_thread_loop_c(mix_t, mix_d, phs_t) {
        /* Loop over sub-threads in super thread */
        sub_thread_loop(sub_t, mix_t, p_d_idx) {
            sub_d = DOMAIN_SUB_DOMAIN(mix_d, p_d_idx); /* Get sub-domain */
            p_d_id = DOMAIN_ID(sub_d); /* Get sub-domain ID */
            Message("\nThis is phase domain: %d; %d\n", p_d_id, p_d_idx);
            if (p_d_id == WATER)
                T = 400.;
            else if (p_d_id == SAND)
                T = 2000.;
            else
                T = 600.;
            if (NNULLP(THREAD_STORAGE(sub_t, SV_T)) &&
                    NNULLP(THREAD_STORAGE(sub_t, SV_VOF))) {
                begin_c_loop(c, sub_t) {
                    C_T(c, sub_t) = T * C_VOF(c, sub_t);
                } end_c_loop(c, sub_t);
            }
        }
    }
    return n_sub_domains;
}

int set_phase_temp3(Domain *mix_d)
{
    Domain *sub_d;
    Thread *mix_t, *sub_t, **phs_t;
    cell_t c;
    int p_d_idx, p_d_id, d_index, n_phases;
    real T;

    /* Get number of sub-domains in mixture */
    n_phases = DOMAIN_N_DOMAINS(mix_d);

    /* Loop over cell thread in mixture domain */
    mp_thread_loop_c(mix_t, mix_d, phs_t) {
        /* Loop over phases */
        phase_loop(p_d_idx, n_phases) {
            sub_d = DOMAIN_SUB_DOMAIN(mix_d, p_d_idx); /* Get sub-domain */
            p_d_id = DOMAIN_ID(sub_d); /* Get sub-domain ID */
            sub_t = phs_t[p_d_idx]; /* Get sub-thread */
            Message("\nThis is phase domain: %d; %d\n", p_d_id, p_d_idx);
            if (p_d_id == WATER)
                T = 600.;
            else if (p_d_id == SAND)
                T = 3000.;
            else
                T = 800.;
            if (NNULLP(THREAD_STORAGE(sub_t, SV_T)) &&
                    NNULLP(THREAD_STORAGE(sub_t, SV_VOF))) {
                begin_c_loop(c, sub_t) {
                    C_T(c, sub_t) = T * C_VOF(c, sub_t);
                } end_c_loop(c, sub_t);
            }
        }
    }
    return n_phases;
}

DEFINE_ON_DEMAND(set_temp)
{
    Domain *mix_d = Get_Domain(1);
    int n_p = 0;
    int method = RP_Get_Integer("mph/method");

    switch (method) {
    case 1:
        n_p = set_phase_temp1(mix_d);
        break;

    case 2:
        n_p = set_phase_temp2(mix_d);
        break;

    case 3:
        n_p = set_phase_temp3(mix_d);
        break;

    default:
        Error("Invalid method!\n");
        break;
    }

    Message("\nThere are %d phases in the mixture\n", n_p);
}
```

例四

```
/*
  * DPM UDF Example
  *
  * Copyright Fluent China, 2005.
  * All Rights Reserved
  *
  */
#ifndef _CHARGE_UDF_H
#define _CHARGE_UDF_H
```

```
#define CHAR_SIZE 256
#define FL_DOMAIN_ID 1

#define MD_MD 3
#define MV_VEC(a)a[MD_MD]
#define MD_VEC(x,y,z)x,y,z

/* User DPM variables */
enum {
   C_P = 0, /* particle charge */
   C_I,         /* time integral of particle charge */
   N_REQUIRED_DPM
};

/* User defined scalalrs */
enum {
   PHI = 0,
   N_REQUIRED_UDS
};

/* User defined memories */
enum {
   MD_VEC(E_X=0,E_Y,E_Z),
   E_M,
   R_P,
   Q_P,
   N_REQUIRED_UDM    /* 6 */
};

#define SV_PHI      SV_UDS_I(PHI)
#define SV_PHI_G SV_UDSI_G(PHI)

#define SV_UDM      SV_UDM_I

#define C_PHI(c,t)      C_UDSI(c,t,PHI)        /* electric potential field */
#define C_PHI_G(c,t) C_UDSI_G(c,t,PHI)    /* potential field gradient */

#define C_EX(c,t) C_UDMI(c,t,E_X)             /* electric field intensity (V/m) */
#define C_EY(c,t) C_UDMI(c,t,E_Y)
#define C_EZ(c,t) C_UDMI(c,t,E_Z)
#define C_EE(c,t) C_UDMI(c,t,E_M)

#define C_R_P(c,t) C_UDMI(c,t,R_P)           /* particulate phase concentration    */
#define C_Q_P(c,t) C_UDMI(c,t,Q_P)           /* particulate phase charge density */


int set_dpm_vars(void);
void get_charge_parameters(void);

extern Domain *fl_domain;
extern real epsilon_0;
extern real d_charge;
extern real charge_max;
```

```
#endif /* _CHARGE_UDF_H */


/*
 * DPM UDF Example
 *
 * Copyright Fluent China, 2005.
 * All Rights Reserved
 *
 */
#include "udf.h"
#include "surf.h"
#include "charge_udf.h"

int set_dpm_vars(void)
{
    int n_user_dpm = RP_Get_Integer("udf/dpm/n-reals");
    char szString[CHAR_SIZE];

    if (n_user_dpm < N_REQUIRED_DPM) {
        sprintf(szString, "Not enough user dpm scalars (Requires %d)", N_REQUIRED_DPM);
        Internal_Error(szString);
        return -1;
    }

    if (NULLP(user_particle_vars)) Init_User_Particle_Vars();

    strcpy(user_particle_vars[C_P].name, "charge");
    strcpy(user_particle_vars[C_P].label, "Charge");
    strcpy(user_particle_vars[C_P].units, "C");
    user_particle_vars[C_P].index = PARTICLE_VARIABLE_MAX + C_P;

    return 0;
}

/*
 * Initialise DPM injection
 */
DEFINE_DPM_INJECTION_INIT(part_dpm_init, injection)
{
    /* Get the latest model parameters */
    get_charge_parameters();
}

/*
 * Update particle charge
 */
DEFINE_DPM_SCALAR_UPDATE(part_dpm_charge, c, t, init, tp)
{
    cphase_state_t *cphase = &(tp->cphase);
    real dq = 0., q0, q_max;
    int i;
    if (init) {
        for (i=0;i<N_REQUIRED_DPM;i++)
            tp->user[i] = 0.;
    }
    else {
        /* Calculate particle charge... */
        q0 = tp->user[C_P];
        q_max = M_PI*epsilon_0*charge_max*C_EE(c,t)*P_DIAM(tp)*P_DIAM(tp);
        if (q0 >= q_max) {
            dq = 0.;
        }
        else {
            dq = d_charge;
        }
        tp->user[C_P] += dq*tp->time_step;

        /* Integrate particle charge... */
        tp->user[C_I] += tp->user[C_P]*tp->time_step;
    }
}

/*
  * Update DPM related Eulerian variables
  */
DEFINE_DPM_SOURCE(part_dpm_var, c, t, Source, strength, tp)
{
    int i;
    real m_ave = 0.;

    /* Particle mass... */
    m_ave = (P_MASS(tp)+P_MASS0(tp))/2;

    /* Particulate phase concentration */
    C_R_P(c, t) += strength*m_ave*(P_TIME(tp)-P_TIME0(tp));


    /* Particulate phase chrage density... */
    C_Q_P(c, t) += strength*tp->user[C_I];

    /* Particle is about to leave the cell, so clear the particle scalar places... */
    /* Do not clear the charge the particle carries! (index 0) */
    for (i=1;i<N_REQUIRED_DPM;i++)
        tp->user[i] = 0.;
}

/*
  * Update User DPM body force
  */
DEFINE_DPM_BODY_FORCE(part_dpm_force, tp, i)
{
    Thread *t = RP_THREAD(&(tp->cCell));
    cell_t c = RP_CELL(&(tp->cCell));
    real NV_VEC(E) = {0.}, Fi;

    if (NNULLP(THREAD_STORAGE(t,SV_UDM)))
        NV_D(E, =, C_EX(c,t), C_EY(c,t), C_EZ(c,t));

    Fi = E[i]*tp->user[C_P]/P_MASS(tp);

    return Fi;
}



/*
  * DPM UDF Example
  *
  * Copyright Fluent China, 2005.
  * All Rights Reserved
  *
  */
#include "udf.h"
#include "sg_udms.h"
#include "charge_udf.h"

Domain *fl_domain = NULL;
real epsilon_0 = 8.854e-12; /* (free space) electric permittivity in farads/m */
real d_charge = 0.;
real charge_max = 0.;

/*
  * Get charge parameters (set via Scheme)
  */
void get_charge_parameters(void)
{
    d_charge = RP_Get_Real("part/charge-rate");
    charge_max = RP_Get_Real("part/charge-max");
}

/*
  * Set UDM variable names
  */
int set_udm_vars(void)
{
    if (NULLP(user_memory_vars)) Init_User_Memory_Vars();

    strcpy(user_memory_vars[Q_P].name, "Charge Density");
    strcpy(user_memory_vars[Q_P].units, "");
    user_memory_vars[Q_P].index = Q_P;

    return 0;
}

/*
  * model initialisation
```

```
 */
int initialise_part(Domain *domain)
{
    Thread *t;
    cell_t c;
    face_t f;
    int nReturn = 0, i;

    thread_loop_c(t, domain) {
        if (NNULLP(THREAD_STORAGE(t,SV_PHI))) {
            begin_c_loop(c, t) {
        for (i=0; i<N_REQUIRED_UDS; i++) {
            C_UDSI(c,t,i) = 0.0;
        }
            } end_c_loop(c, t);
        }

        if (NNULLP(THREAD_STORAGE(t, SV_UDM))) {
            begin_c_loop(c, t) {
        for (i=0; i<N_REQUIRED_UDM; i++) {
            C_UDMI(c,t,i) = 0.0;
        }
            } end_c_loop(c, t);
        }
    }

    return nReturn;
}

void charge_initialisation(Domain *domain)
{
    int ns = 0;
    char szString[CHAR_SIZE];

    /* Make sure there are enough user defined scalars and memories. */
    if (n_uds < N_REQUIRED_UDS) {
        sprintf(szString, "Not enough user defined scalars (Requires %d)", N_REQUIRED_UDS);
        Internal_Error(szString);
    }
    if (n_udm < N_REQUIRED_UDM) {
        sprintf(szString,    "Not    enough    user    defined    memories    (Requires    %d)",
N_REQUIRED_UDM);
        Internal_Error(szString);
    }

    ns = initialise_part(domain);
    if (ns != 0)
        Internal_Error("Error: Cannot initialise model!");

    ns = set_udm_vars();
    if (ns != 0)
        Internal_Error("Error: Cannot set UDM user variables!");
    if (sg_dpm) {
        ns = set_dpm_vars();
        if (ns != 0)
            Internal_Error("Error: Cannot set DPM user variables!");
    }
    get_charge_parameters();
}

/*
  * Initialise DPM related Eulerian variables
  */
void initialise_dpm(Domain *domain)
{
    Thread *t;
    cell_t c;

    thread_loop_c(t, domain) {
        if (NNULLP(THREAD_STORAGE(t,SV_UDM))) {
            begin_c_loop(c, t) {
        C_R_P(c,t) = 0.0;
        C_Q_P(c,t) = 0.0;
            } end_c_loop(c, t);
        }
    }
}
/*
  * Update DPM related Eulerian variables
  */
void update_dpm(Domain *domain)
{
    Thread *t;
    cell_t c;

    thread_loop_c(t, domain) {
        if (NNULLP(THREAD_STORAGE(t,SV_UDM))) {
            begin_c_loop(c, t) {
        C_R_P(c,t) /= C_Volume(c, t);
        C_Q_P(c,t) /= C_Volume(c, t);
            } end_c_loop(c, t);
        }
    }
}

/*
  * Update electric field intensity (E) (V/m)
  */
void update_E(Domain *domain)
{
    Thread *t;
    cell_t c;
    real dPhi[ND_ND] = {0.};

    thread_loop_c(t, domain) {
        if (NNULLP(THREAD_STORAGE(t, SV_PHI_G)) &&
        NNULLP(THREAD_STORAGE(t, SV_UDM))) {
            begin_c_loop(c,t) {
        /* Get dPhi() */
        NV_V(dPhi, =, C_PHI_G(c,t));
        ND_VEC(C_EX(c, t) = -dPhi[0],
                    C_EY(c, t) = -dPhi[1],
                    C_EZ(c, t) = -dPhi[2]);
        C_EE(c, t) = NV_MAG(dPhi);
            } end_c_loop(c,t);
        }
    }
}

/*
 * model general initialisation
 */
DEFINE_INIT(charge_init, domain)
{
    charge_initialisation(domain);
}

/*
 * Adjust routine for initial setup
 */
DEFINE_ADJUST(charge_adjust, domain)
{
    int current_iter = (nres == 0) ? (0) : ((int)count2[nres-1]);
    int dpm_niter = RP_Get_Integer("dpm/niter");
    static int step_iteration;
    cxboolean dpm_update = FALSE, dpf_update = FALSE;
    char szString[CHAR_SIZE];

    /* Make sure there are enough user defined scalars and memories. */
    if (n_uds < N_REQUIRED_UDS) {
        sprintf(szString, "Not enough user defined scalars (Requires %d)", N_REQUIRED_UDS);
        Internal_Error(szString);
    }
    if (n_udm < N_REQUIRED_UDM) {
        sprintf(szString, "Not enough user defined memories (Requires %d)",
N_REQUIRED_UDM);
        Internal_Error(szString);
    }

    get_charge_parameters();

    if (rp_unsteady && first_iteration)
        step_iteration = 0;

    if (rp_unsteady)
        current_iter = ++step_iteration;

    /*
```

```
     * Update variables
     */

    /* Initialise DPM related variables... before the start of DPM iteration */
    if (sg_dpm &&
        (dpm_niter && current_iter && (((current_iter+1) % dpm_niter) == 0))) {
      /*
      CX_Message("Init DPM: Iter=%d, mode=%d \n",current_iter,(current_iter % dpm_niter));
      */
      initialise_dpm(domain);
    }

    /* Update some variables after DMP iteration */
    dpm_update = (dpm_niter && current_iter && ((current_iter % dpm_niter) == 0));
    if (sg_dpm && dpm_update)
      update_dpm(domain);

    /* Update variables */
    /* electric field E */
    update_E(domain);
}

/*
 * UDS equation diffusion term
 */
DEFINE_DIFFUSIVITY(phi_diffusivity, c, t, ns)
{
    real diff = 0.;

    if (ns == PHI)
      diff = 1.;
    else
      diff = 0.;

    return diff;
}

/*
 * Source terms for electric field equations
 */
DEFINE_SOURCE(phi_emf_source, c, t, dS, eqn)
{
    int ns = eqn - EQ_UDS;    /* user scalar index */
    int i;
    real source = 0.;

    dS[eqn] = 0.0;

    if (ns != PHI) return source;

    if (NNULLP(THREAD_STORAGE(t, SV_UDM)))
      source = C_Q_P(c, t)/epsilon_0;
    return source;
}
```

例五

```
#include "udf.h"

/*
  * Constant Parameters
  */
#define N_WALLS 15 /* Total number of walls to consider */
#define RR 6
#define VP 3.5
#define TW 25.0 /* Water temperature */
#define SMALL_T 1.0

/* Wall Parameter Lists */
/* wall name must coorespond to its ID, sied type, roler number, roler diameter, etc */
char *w_name[] = {"zugun", "zugun-in", "zugun-out", "sec1a", "sec1a-in", "sec1a-out", \
                  "sec1b", "sec1b-in", "sec1b-out", "sec2", "sec2-in", "sec2-out", \
                  "aircool", "aircool-in", "aircool-out"};
int    w_ID[] = {24, 26, 25, 21, 23, 22, 18, 20, 19, 15, 17, 16, 12, 14, 13};
int    w_side[] = {0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2};
int    w_n_roler[] = {0, 1, 1, 0, 6, 6, 0, 7, 7, 0, 5, 5, 0, 5, 5};
real w_d_roler[] = {0.,0.08,0.08,0.,0.13,0.13,0.,0.13,0.13,0.,0.13,0.13,0.,0.13,0.13};
real w_W[N_WALLS] = {270.,270.,270.,364.,364.,364.,240.,240.,240.,128.,128.,128.,0.,0.,0.};
real w_L[N_WALLS] = {0.};
real w_A[N_WALLS] = {0.};
real w_T[N_WALLS] = {0.};
real w_Q[N_WALLS] = {0.};

/*
  * Get index of the specified wall thread in the wall list
  */
int get_wall_index(int t_id)
{
    int w_idx = -1, i;

    for (i = 0; i < N_WALLS; i++) {
        if (t_id == w_ID[i]) {
            w_idx = i;
            break;
        }
    }
    return w_idx;
}

/*
  * Calculate Averaged Wall Parameters:
```

```
 *      Area, Mean Temperature, Mean Distance to Entry Point
 */
int calc_wall_para(Thread *t)
{
#if !RP_HOST
    int t_id, w_idx, i;
    real sum, sum_a, sum_f, sum_l;
    real    y, z, l, T_avg;
    face_t f;
    real A[ND_ND], x[ND_ND], area, T;

    /* locate the thread in the wall thread ID list */
    t_id = THREAD_ID(t);
    w_idx = get_wall_index(t_id);
    if (w_idx != -1) {
        /* found the thread ID in the list.
              calculate the area weighted mean temperature */
        sum = sum_a = sum_f = sum_l = 0;
        begin_f_loop(f, t) {
            F_CENTROID(x, f, t);
            F_AREA(A, f, t);
            area = NV_MAG(A);
            T = F_T(f, t) - 273.;
            sum_a += area;
            sum += (area*T);
            sum_f += 1.;
            /* calc. distance to level */
            y = x[1];
            z = x[2];
            l = RR*asin(fabs(z)/RR);
            sum_l += l;
        } end_f_loop(f, t);
#if RP_NODE
        PRF_GRSUM4(sum, sum_a, sum_l, sum_f);
#endif /* RP_NODE */
        T_avg = sum / sum_a;
        w_T[w_idx] = T_avg;
        w_A[w_idx] = sum_a;
        w_L[w_idx] = sum_l / sum_f + 0.4;

  }

    return w_idx;
#endif /* !RP_HOST */
}

/*
 * Calculate Total Heat Transfer on each Wall
 */
real calc_wall_heat(int w_idx)
{
    int i = w_idx;
    real two_a = 0., r, alpha, h, w_q;
    real Q1 = 0., Q2 = 0., Q3 = 0., Q4 = 0.;

    /* 1: Roler heat transfer */
    if (w_n_roler[i] != 0) {
        if (w_L[i] < 4.) {
            two_a = -0.3116 + 4.6105 * w_L[i];
        }
        else {
            two_a = -0.3116 + 4.6105 * 4.;
        }
        Q1 = w_n_roler[i]*(two_a*w_d_roler[i]*0.5/360.*11513.7*pow(w_T[i], 0.7556)*
                        pow(VP, -0.2010)*pow(two_a, -0.1639));
    }
    else {
        two_a = 0.;
        Q1 = 0.;
    }
    /* 2: Water cooling */
    if (w_side[i] != 0) {
        w_q = w_W[i]/60./2./w_A[i];
        if (w_T[i] > 900.) {
            h = 1.095e15*pow(w_T[i], -4.15)*pow(w_q, 0.75);
        }
        else if (w_T[i] > 500. && w_T[i] <= 900.) {
            h = 3.78e6*pow(w_T[i], -1.34)*pow(w_q, 0.785);
        }
        else {
            h = 3.78e6*pow(500., -1.34)*pow(w_q, 0.785);
        }
        Q2 = w_A[i]*h*(w_T[i] - TW);
    }
    else {
        Q2 = 0.;
    }
    /* 3: Air cool and radiation */
    Q3 = w_A[i]*0.8*5.67e-8*(pow((w_T[i]+273), 4.) - pow((TW+273), 4.));
    /* 4: Water evaporation */
    if (w_side[i] == 0 || w_side[i] == 1) {
        alpha = 0.39;
    }
    else {
        alpha = 0.;
    }
    Q4 = alpha/(1.-alpha)*(Q1 + Q2 + Q3);

    /* Total heat (excluding radiation) */
    w_Q[i] = Q1 + Q2 + Q4;

    return two_a;
}

/*
 * Calculate and Display Total Heat Transfer, Heat Transfer Coefficient for each Wall
```

```
  */
DEFINE_ON_DEMAND(cc_calc_heat)
{
#if !RP_HOST
   Domain *domain = Get_Domain(1);
   Thread *t;
#endif /* !RP_HOST */
   int i;
   real two_a;

   if (!Data_Valid_P()) return;

#if !RP_HOST
   /* Calculate mean wall parameters (area, temperature,...) */
   thread_loop_f(t, domain) {
      if (BOUNDARY_FACE_THREAD_P(t)) {
         i = calc_wall_para(t);
      }
   }
#endif /* !RP_HOST */

   /* Calculate and display total heat transfer for each zone surface */
   Message0("\t\t L\t A\t    2a\t    T\t\tQ\t        h\n");
   for (i = 0; i < N_WALLS; i++) {
      /* Total heat (excluding radiation) */
      two_a = calc_wall_heat(i);

      Message0("%s        \t%5.2f\t%6.4f\t%7.4f\t%7.2f\t%10.1f\t%8.1f\n",
                 w_name[i], w_L[i], w_A[i], two_a, w_T[i], w_Q[i],
                 (w_Q[i]/(w_A[i]*(w_T[i]-TW))));
   }
}

DEFINE_PROFILE(cc_heat_tran, t, pos)
{
#if !RP_HOST
   Thread *t0 = THREAD_T0(t);
   cell_t c0;
   face_t f;
   int w_idx;
   real two_a, h, q, T_w;

   /* Calculate averaged wall parameters, and get wall index */
   w_idx = calc_wall_para(t);

   if (w_idx != -1) { /* Check wall index */
      /* Calculate wall total heat transfer */
      two_a = calc_wall_heat(w_idx);
      /* Calculate wall heat flux */
      q = w_Q[w_idx]/w_A[w_idx];
      /* Calculate effective wall heat transfer coefficient */
      /* method 1: use this for all faces on the thread */
      h = w_Q[w_idx]/(w_A[w_idx]*(w_T[w_idx]-TW));
```

```
        /* Loop over all face on wall thread */
        begin_f_loop(f, t) {
            T_w = F_T(f, t) - 273.;
            /* method 2: calculate h for each face */
            /*
            h = q/MAX(SMALL_T, (T_w - TW));
            */
            /* Assign heat transfer coefficient to F_PROFILE */
            F_PROFILE(f, t, pos) = h;
            /* Store heat transfer coefficient on a User Defined Memory */
            if (NNULLP(THREAD_STORAGE(t0, SV_UDM_I)) && sg_udm != 0) {
        c0 = F_C0(f, t);
        C_UDMI(c0, t0, 0) = -h;
            }
        } end_f_loop(f, t);
    }
#endif /* !RP_HOST */
}
```