Skript der Vorlesung

"Parallele Programmierung mit der PS/3"

Unix-Befehle

cat Anzeige des Inhaltes einer Datei
chmod u+x script Änderung des Execute Attribut (Starten ohne Prefix ksh)
chmod Change Modus einer Datei (chmod u+x datei)
clear Löschen des Bildschirms

df Anzeige des freien Festplattenspeichers
du Anzeige des benutzten Festplattenspeichers

grep Suchen von Zeichenfolgen in Dateien

ksh script Starten des Scriptes „script“
ksh Aufruf der Kornshell

ls Anzeige der Dateien (-p, -l)

mkdir Anlegen eines Verzeichnis

ps Anzeige des Prozessstatus
pwd Print working Directory (Aktuelle Verzeichnis)

rmkdir Löschen eines Verzeichnis

sort Sortieren einer Datei

wc Word Count
who Anzeioge des aktuellen Benutzers

vi-Editor

Text bearbeiten

Befehl	**Beschreibung**
`ESC i`	Fügt Text vor dem aktuellen Zeichen ein
`ESC l`	Fügt Text am Zeilenanfang ein
`ESC a`	Fügt Text nach dem aktuellen Zeichen ein
`ESC A`	Fügt Text am Zeilenende hinzu
`ESC o`	Fügt eine Zeile unter der aktuellen Zeile ein
`ESC O`	Fügt eine Zeile oberhalb der aktuellen ein
`ESC R`	Eingabe von Text im Überschreibemodus

Löschen von Zeichen

Befehl	Beschreibung
ESC x	Löscht ein **Zeichen**
ESC dw	Löscht bis Wortende
ESC db	Löscht bis zum Wortanfang
ESC dnw	Löscht die nächsten n Wörter
ESC dd	Löscht die Zeile
ESC d0	Löscht bis zum Zeilenanfang
ESC D	Löscht bis zum Zeilenende
ESC dg	Löscht bis zur letzten Zeile
ESC dlg	Löscht bis zur ersten Zeile

Es existieren eine Anzahl von Zwischenspeichern, die man zum Kopieren und Einfügen von Texten verwenden kann. Der Standardpuffer, der immer verfügbar ist, und die Puffer a-z können beliebig verwendet werden.

Zwischenspeicher

Befehl	Beschreibung
ESC yy	Übernimmt eine Textzeile in den Standardpuffer
ESC nyy	Übernimmt die nächsten n Zeilen in den Standardpuffer
ESC yw	Übernimmt das nächste Wort in den Standardpuffer
ESC ynw	Übernimmt die nächsten n Wörter in den Standardpuffer
ESC dd	Löscht die aktuelle Zeile und speichert diese in den Standardpuffer
ESC ndd	Löscht die nächsten n Zeilen und speichert diese in den Standardpuffer
ESC p	Wenn der Standardpuffer eine oder mehrere Zeilen enthält, werden diese unterhalb der aktuellen Zeile eingefügt
ESC P	Wenn der Standardpuffer eine oder mehrere Zeilen enthält, werden diese oberhalb der aktuellen Zeile eingefügt

Datei-Operationen

Befehl	Beschreibung
ESC ZZ	Speichert und beendet
ESC :w name	Schreibt den Inhalt in die Datei "name"
ESC :w	Speichert den Inhalt
ESC :x	Speichert die Änderung und beendet
ESC :q!	Ende ohne zu speichern
ESC :q↵	Ende, nur wenn keine Änderung

Programmoptionen des gcc

Beispiel	**Beschreibung**
gcc bsp1.c	Übersetzt und linkt die Datei Ausgabe in a.out (Assember.out)
./bsp1	Startet das Programm
gcc bsp1.c -o bsp1	Übersetzt und linkt die Datei Ausgabe in bsp1
gcc -c bsp1.c	Übersetzt die Datei bsp1.c und erstellt die Datei bsp1.o
gcc -o bsp bsp1.c bsp2.c	Übersetzt die Datei bsp1.c und bsp2.c erstellt die Datei bsp1.o und bsp2.o Linkt beide *.o zu einer "Exe-Datei" namens bsp
ppu32-gcc	PS3 Power PC Komplier/Linker
ppu-gcc	PS3 Power PC Komplier/Linker
spu-gcc	PS3 SPU Komplier/Linker
spu32-gcc	PS3 SPU Komplier/Linker
-m32 -Wl,-melf32ppc	Setzt den Kompiler in den 32-Bit Modus
-lm	Lädt zusätzlich die Library libm
-lspe	Lädt zusätzlich die Library libspe
-IPath	Setzt einen zusätzlichen Pfad für die include-Dateien
-LPath	Setzt einen zusätzlichen Pfad für die Bibliotheken
-O	Optimiert die Übersetzung gcc -O -o bsp bsp1.c
-g	Fügt debug-Code hinzu gcc -g -o bsp bsp1.c gcc -O -g -o bsp bsp1.c
.C	Endung für C++-Dateien
.cc	Endung für C++-Dateien
g++	Aufruf des C++ Kompiler

kill -9 prozessnr löscht den Prozess

Threads / Mutex

Funktion: pthread_create

Includedatei: #include <pthread.h>

int pthread_create(
pthread_t *thread, Thread description record, int
const pthread_attr_t *attr, NULL
void *(*start_routine)(void*), Pointer zur Prozedur
void *arg Agrumente,meist eine struct
);

Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *print_char (void *ch) {
  printf ("%s", (char*)ch);
  return NULL;
}

int main () {
  char ch1='-', ch2='*';
  pthread_t p1, p2;

  if ( pthread_create (&p1, NULL, print_char, &ch1) ) {
     fprintf (stderr, "Fehler bei 1. Thread\n");
     exit (EXIT_FAILURE);
  }
  if ( pthread_create (&p2, NULL, print_char, &ch2) ) {
     fprintf (stderr, "Fehler bei 2. Thread\n");
     exit (EXIT_FAILURE);
  }

  pthread_join (p1, NULL);
  pthread_join (p2, NULL);

  puts ("Ende");
  return 0;
}
```

```
gcc -D_REENTRANT bsp1.c -lpthread
```

Die Option Reentrant sichert threadsichere Funktion zu.

Mutex

```
pthread_mutex_t   sumMutex;                    // Definition

pthread_mutex_init(&sumMutex, NULL);           // Init

pthread_mutex_lock(&sumMutex);                 // Sperren
pthread_mutex_unlock(&sumMutex);               // Freigeben
```

Zeitmessung

Mit folgendem Programmcode kann man die Laufzeit messen:

```
struct timeval t1,t2;
double d1,d2;

gettimeofday(&t1,NULL);
   // calc
gettimeofday(&t2,NULL);
d1 = t1.tv_sec + (t1.tv_usec/1000000.0);
d2 = t2.tv_sec + (t2.tv_usec/1000000.0);
printf("Zeit %f\n",d2-d1);
```

Debugger

Optionen des Debuggers:

list oder l	Listing des Quellcodes um den aktuellen Punkt
l 1,5	Listing von Zeile bis 5
run oder r	Starten des Programms
break zeile	Haltepunkt setzen
break zeile if Bedingung	Haltepunkt setzen
step oder s	Einzelschritt, in eine Funktion
next oder n	Schritt,
continue oder c	Programm weiterlaufen lassen
info breakpoints	Anzeige der Haltepunkte
delete 2	Löschen des 2. Haltepunkts
print i	Anzeige des Inhalts von i
print	
whatis i	Anzeige des Typs von i
set variable s=0	ändern der Summe
quit oder q	Ende
Enter	letzten Befehl wiederholen

Vectorfelder SIMD

In einem Register passen vier float-Zahlen. Mittels spezieller Befehle können nun einfache Operationen parallel ausgeführt werden. Damit zeigen sie jeweils auf den Anfang der einzelnen Felder. Durch die "union" kann man auf die einzelnen beiden Varianten, entweder als Feld, oder als Register, zugreifen. Durch die Pointer ist nun die Weiterschaltung sehr elegant und schnell.

Einfacher Beispielcode für allgemeine Vectorfelder:

```
  vector float vA={1,2,3,4};
  vector float vB={3,4,5,6};
  vector float vC={0,0,0,0};

  vC = spu_add(vA,vB);
oder
  vC = vA +  vB;
  print_f_vector("vA",vA);
  print_f_vector("vB",vB);
  print_f_vector("vC",vC);        // 4,6,8,10

  vC = spu_mul(vA,vB);
oder
  vC = vA *  vB;
  print_f_vector("vA",vA);
  print_f_vector("vB",vB);
  print_f_vector("vC",vC);        // 3,8,15,24
```

Pointer:

```
// Diese union sichert den Zugriff auf den Speicher mittels zweier verschiedener Adressierungen
// 1. als normale Pointer, beispielsweise 10000 Zahlen
// 2. als VectorFeld, jeweils genau VIER float Zahlen
typedef union {
  float *fVal;
  vector float * myVec;
} floatVec;
```

Quellcode der 2. Variante

```
float *a, *b, *c, *d;        // Pointer
floatVec a4a, b4a, c4a, d4a;
     // root Referenz zuweisen
for (k=0; k<N; k++) {
  for (i=0; i<MAX; i+=4) {
   a4a.myVec = a4a.myVec * a4a.myVec;
     …
   d4a.myVec =  a4a.myVec + b4a.myVec + c4a.myVec;
     // Increment
}
```

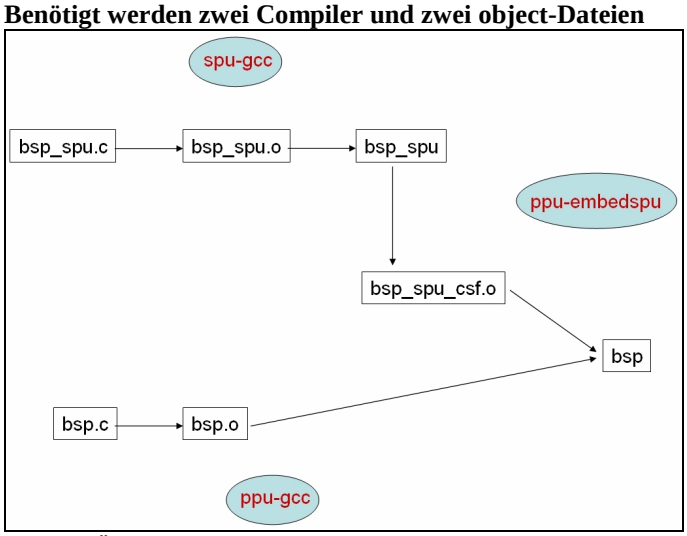
Spu-Funktionen

Funktion	**Beschreibung**
d = spu_add(a, b)	Vector add, d=a+b
d = spu_addx(a, b, c)	Vector add extended
d = spu_genb(a, b)	Vector generate borrow
d = spu_genbx(a, b, c)	Vector generate borrow extended
d = spu_genc(a, b)	Vector generate carry
d = spu_gencx(a, b, c)	Vector generate carry extended
d = spu_madd(a, b, c)	Vector multiply and add; d=a*b+c
d = spu_mhhadd(a, b, c)	Vector multiply high high and add
d = spu_msub(a, b, c)	Vector multiply and subtract; d=a*b-c
d = spu_mul(a, b)	Vector multiply, d=a*b
d = spu_mulh(a, b)	Vector multiply high
d = spu_mulhh(a, b)	Vector multiply high high
d = spu_mulo(a, b)	Vector multiply odd
d = spu_mulsr(a, b)	Vector multiply and shift right
d = spu_nmadd(a, b, c)	Negative vector multiply and add
d = spu_nmsub(a, b, c)	Negative vector multiply and subtract
d = spu_re(a)	Vector floating-point reciprocal estimate
d = spu_rsqrte(a)	Vector floating-point reciprocal square root estimate
d = spu_sub(a, b)	Vector subtract; d=a-b
d = spu_subx(a, b, c)	Vector subtract extended

Hinweis:
Komplexere Befehle, wie sqrt, sin, werden nacheinander abgehandelt !

Threads mit spu

Benötigt werden zwei Compiler und zwei object-Dateien



System-Überblick

struct.h

```
//#include <stdint.h>

typedef struct  {
  int id;
  int n;
} dma_packet;
```

Threadserzeugung in ppc-main:

```
#include "struct.h"
extern spe_program_handle_t   calculate_spu;

  dma_packet cb[ N]   __attribute__((aligned(16)));
  speid_t spe_ids[ N];
  pthread_t p1,p2;

for (i=0..n-1) {
  spe_ids[i] = spe_create_thread(0, &calculate_spu, &cb[i] , NULL, -1, 0);
  if (spe_ids[i] == 0) {
      printf("Failed spu_create_thread(errorcode: =%lld)\n",  spe_ids[i]);
      return;
   }
}

for (i=0..n-1) {
    spe_wait(spe_ids[i], NULL, 0);  // Join
}
```

Threads in spu:

```
#include "pi.h"

int main(unsigned long long speid, unsigned long long param, unsigned long long envp) {
    int i,n;
    int tag_id;
    dma_packet cb  __attribute__((aligned(16)));       // warum ein alignment von 16 Byte

    // daten holen
    mfc_get(&cb, param, sizeof(cb), tag_id, 0 ,0);
    if (tag_id>0)   printf("tag id: %d\n",tag_id);
    mfc_write_tag_mask(1 << tag_id);  // warten
    mfc_read_tag_status_any();  // any  statt all

    n = cb.n;
    // Aktion

    // Zurück schreiben
    mfc_put(&cb, param, sizeof(dma_packet), tag_id, 0 ,0);
    mfc_write_tag_mask(1 << tag_id);  // warten
    mfc_read_tag_status_any();  // any  statt all
```