```
1    // Handout_week7.b
2    //   code copied from “https://github.com/Keitokuch/linux-4.15-lb”


3    #ifndef _SCHED_JC_MLP_H
4    #define _SCHED_JC_MLP_H
5
6    #include <linux/sched.h>
7
8    extern int is_jc_sched;
9
10   struct jc_lb_data {
11       int src_non_pref;
12       int delta_hot;
13       int cpu_idle;
14       int cpu_not_idle;
15       int cpu_newly_idle;
16       int same_node;
17       int prefer_src;
18       int prefer_dst;
19       int src_len;
20       unsigned long src_load;
21       unsigned long dst_load;
22       int dst_len;
23       long delta_faults;
24       int extra_fails;
25       int buddy_hot;
26       unsigned long total_faults;
27   };
28
29   int jc_mlp_main(struct jc_lb_data *data);
30
31   #ifdef CONFIG_JC_SCHED_FXDPT
32   typedef s32 fxdpt_t;
33   typedef s64 fxdpt_ext;
34
35   #define FXDPT_FBITS 11
36
37   /* Can't left shift negative int */
38   /* Add 0.5/-0.5 to round to nearest int  */
39   #define FXDPT_ONE ((fxdpt_t)((fxdpt_t)1 << FXDPT_FBITS))
40   #define float_to_fxdpt(F) ((fxdpt_t)((F) * FXDPT_ONE + ((F) >= 0? 0.5 : -0.5)))
41   #define int_to_fxdpt(F) ((fxdpt_t)((F) * FXDPT_ONE))
42   /* Extend to double length to avoid overflow */
43   #define fxdpt_mul(A, B)         \
44       ((fxdpt_t)(((fxdpt_ext)(A) * (fxdpt_ext)(B)) >> FXDPT_FBITS))
45   #define fxdpt_div(A, B)                 \
46       ((fxdpt_t)(((fxdpt_ext)(A) * FXDPT_ONE) / (B)))
47
48   #define tofloat(T) ((float) ((T)*((float)(1)/(float)(1L << FXDPT_FBITS))))
49
50   #define dtype               fxdpt_t
51   #define ftodtype(F)         float_to_fxdpt(F)
52   #define itodtype(F)         int_to_fxdpt(F)
53   #define todtype(F)          itodtype(F)
54   #define dtype_mul(A, B)     fxdpt_mul(A, B)
55   #define dtype_div(A, B)     fxdpt_div(A, B)
56   #else
57   #define dtype               float
58   #define dtype_mul(A, B)     ((A) * (B))
59   #define dtype_div(A, B)     ((A) / (B))
60   #define ftodtype(F)          ((float)(F))
61   #define itodtype(F)          ((float)(F))
62   #define todtype(F)          ((float)(F))
63   #define tofloat(F)          ((float)(F))
64   #endif  // JC_SCHED_FXDPT
65
66   #endif
67
68
69
70
```



```
71   // Implementation
72
73   #include <linux/kernel.h>
74   #include <asm/fpu/api.h>
75   #include <linux/syscalls.h>
76
77   #include "jc_mlp.h"
78
79
80   #define NR_FEAT     15
81   #define ftod(F)         ftodtype(F)
82
83   dtype w1[] = {
84       ftod(0.565644),  ftod(0.384824), ftod(-0.042528), ftod(-0.264426),  ...
85       ..
86   };
87
88   dtype b1[] = {
89       ftod(-0.257286), ftod(-0.220656), ftod(-0.312300), ftod(0.485314), ...
90   };
91
92   dtype w2[] = { ftod(-0.409050), ftod(-0.578690), ftod(-0.446989), ... }
93
94   dtype b2[] = { ftod(0.405319) };
95
96   #define m2d(x, i, j)    ((x)->values[i * (x)->ncol + j])
97   #define m1d(x, i)       ((x)->values[i])
98   #define _ReLU(x)        (x > 0 ?  x : 0)
99   #define ftox(f)         (*(unsigned *)&((float){f}))
100
101  int is_jc_sched = 0;
102  EXPORT_SYMBOL(is_jc_sched);
103
104  struct matrix {
105      int nrow;
106      int ncol;
107      dtype *values;
108  };
109
110  static inline void matmul(struct matrix *X, struct matrix *Y, struct matrix *Z)
111  {
112      int i, j, k;
113      for(i = 0; i < X->nrow; i++)
114          for(j = 0; j < Y->ncol; j++)
115              for(k = 0; k < X->ncol; k++)
116              {
117                  m2d(Z, i, j) = m2d(Z, i, j) + dtype_mul(m2d(X, i, k), m2d(Y, k, j));
118              }
119  }
120
121  static inline void matadd(struct matrix *X, struct matrix *Y, struct matrix *Z)
122  {
123      int i;
124      for (i = 0; i < X->nrow * X->ncol; i++) {
125          Z->values[i] = X->values[i] + Y->values[i];
126      }
127  }
128
129  static inline void ReLU(struct matrix *X)
130  {
131      int i;
132      for (i = 0; i < X->nrow * X->ncol; i++) {
133          X->values[i] = _ReLU(X->values[i]);
134      }
135  }
136
```

```
137 static int forward_pass(struct matrix *input)
138 {
139     dtype output;
140     dtype o1[10] = {0};
141     dtype o2[10] = {0};
142
143     struct matrix W1 = {NR_FEAT, 10, w1};
144     struct matrix out1 = {1, 10, o1};
145     struct matrix B1 = {1, 10, b1};
146     struct matrix W2 = {10, 1, w2};
147     struct matrix out2 = {1, 1, o2};
148     struct matrix B2 = {1, 1, b2};
149
150     matmul(input, &W1, &out1);
151
152     matadd(&out1, &B1, &out1);
153
154     ReLU(&out1);
155
156     matmul(&out1, &W2, &out2);
157
158     matadd(&out2, &B2, &out2);
159
160     output = m1d(&out2, 0);
161
162     /* printk("forward_pass output: %08x", ftox(tofloat(output))); */
163     return output > ftod(0.5) ? 1 : 0;
164 }
165
166 #ifdef CONFIG_JC_SCHED_FXDPT
167 int jc_mlp_main(struct jc_lb_data *data) {
168     int output;
169     struct matrix input = {1, NR_FEAT, NULL};
170     dtype delta_faults;
171
172     if (data->total_faults)
173         delta_faults = dtype_div(itodtype(data->delta_faults), itodtype(data->total_faults));
174     else
175         delta_faults = itodtype(data->delta_faults);
176
177     input.values = (dtype[]) {
178         itodtype(data->src_non_pref),
179         itodtype(data->delta_hot),
180         itodtype(data->cpu_idle),
181         itodtype(data->cpu_not_idle),
182         itodtype(data->cpu_newly_idle),
183         itodtype(data->same_node),
184         itodtype(data->prefer_src),
185         itodtype(data->prefer_dst),
186         itodtype(data->src_len) - itodtype(2),
187         dtype_div(itodtype(data->src_load), itodtype(1000)),
188         dtype_div(itodtype(data->dst_load), itodtype(1000)),
189         itodtype(data->dst_len),
190         delta_faults,
191         itodtype(data->extra_fails),
192         itodtype(data->buddy_hot),
193     };
194
195     output = forward_pass(&input);
196
197     return output;
198 }
199 #else
200 int jc_mlp_main(struct jc_lb_data *data) {
201     int output;
202     struct matrix input = {1, NR_FEAT, NULL};
203     dtype delta_faults;
204
205     kernel_fpu_begin();
206
207     if (data->total_faults)
```



```
208         delta_faults = (dtype)data->delta_faults / data->total_faults;
209     else
210         delta_faults = (dtype)data->delta_faults;
211
212     input.values = (dtype[]) {
213         (dtype)data->src_non_pref,
214         (dtype)data->delta_hot,
215         (dtype)data->cpu_idle,
216         (dtype)data->cpu_not_idle,
217         (dtype)data->cpu_newly_idle,
218         (dtype)data->same_node,
219         (dtype)data->prefer_src,
220         (dtype)data->prefer_dst,
221         (dtype)data->src_len - 2,
222         (dtype)data->src_load / 1000,
223         (dtype)data->dst_load / 1000,
224         (dtype)data->dst_len,
225         delta_faults,
226         (dtype)data->extra_fails,
227         (dtype)data->buddy_hot,
228     };
229
230     output = forward_pass(&input);
231
232     kernel_fpu_end();
233
234     return output;
235 }
236 #endif
```