

Where there is life, the  
re is hope

RSS订阅

原

openMP学习笔记

2018年01月21日 14:52:08

阅读数:

2

收藏

评论

微信

微博

QQ

本次学习课程来自Intel 高级研究员Tim Mattson

课程视频下载地址(全英文且无字幕):

链接: <https://pan.baidu.com/s/1nw6pcRv> 密码: aolo

虽然最近量子计算和Nvidia CUDA技术越来越热, 但是工业上都采用arm架构的嵌入式设备, 负担不起nvidia的成本, 所以学enMP、SIMD之类的还是比较重要的。

1.openmp是干什么的

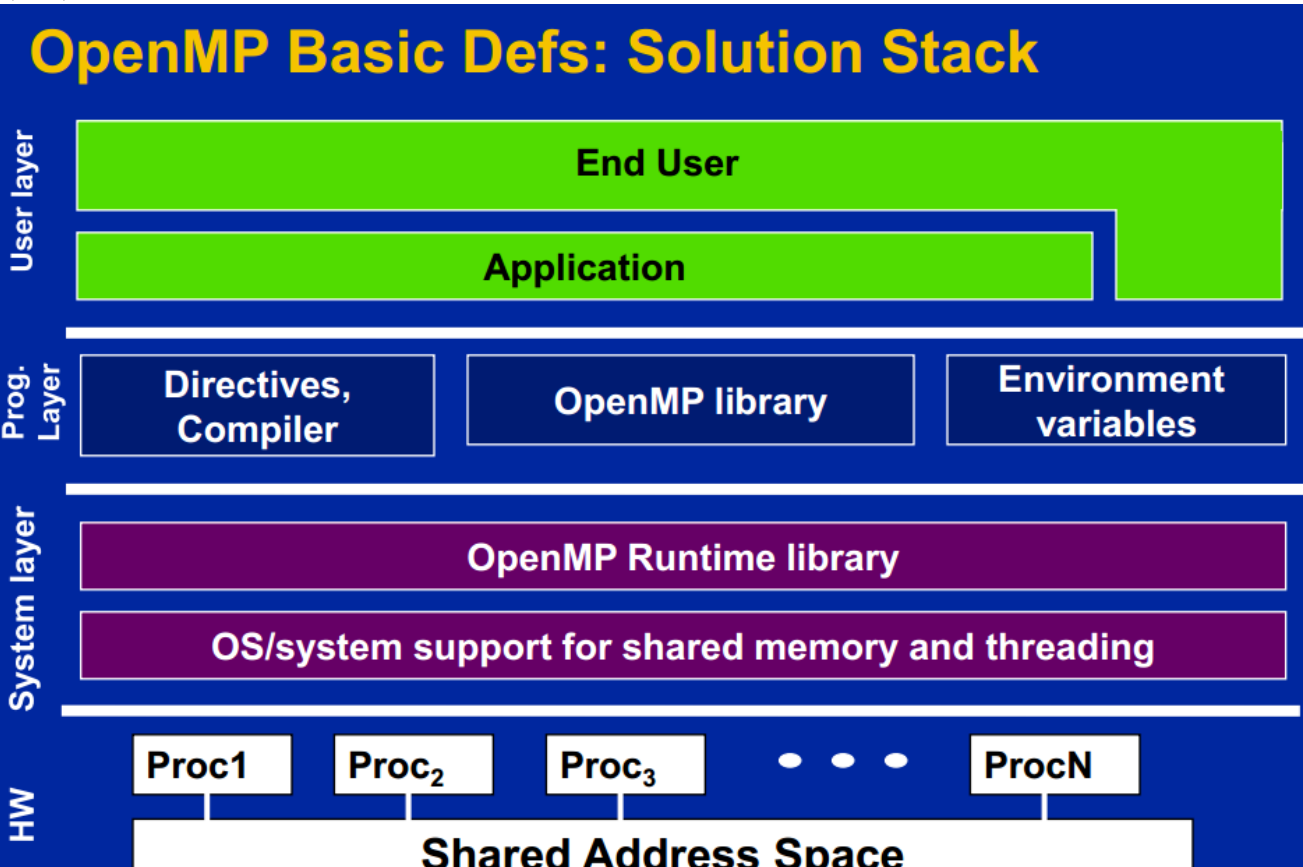
openmp是一个跨平台的、较常用的基于共享内存(地址空间)的并行编程模型, 它提供一组编译指令集和库例程给并行应用开发者, openmp只支持fortran,c和c++. Linux下的gcc天然支持openmp (注意版本)。

[plain]

```
1. echo |cpp -fopenmp -dM |grep -i open
```

```
jst@jst-pc:~$ echo |cpp -fopenmp -dM |grep -i open
#define _OPENMP 201307
jst@jst-pc:~$ http://blog.csdn.net/jinshengtao
```

openmp的架构层次如下图所示



openmp内线程是如何交互的？

openmp是一个多线程、共享地址模型，不同线程通过共享变量交互信息。但是随意的数据共享将会导致竞争问题，即每次程序运行的结果都会因OS不同的线程调度执行次序而改变。为了控制竞争问题，openmp采用同步策略来保护数据冲突。但是，同步策略是非常昂贵的，消耗performance的。因此，合格的openmp程序员，将以最小的同步代价来制定数据的访问规则。

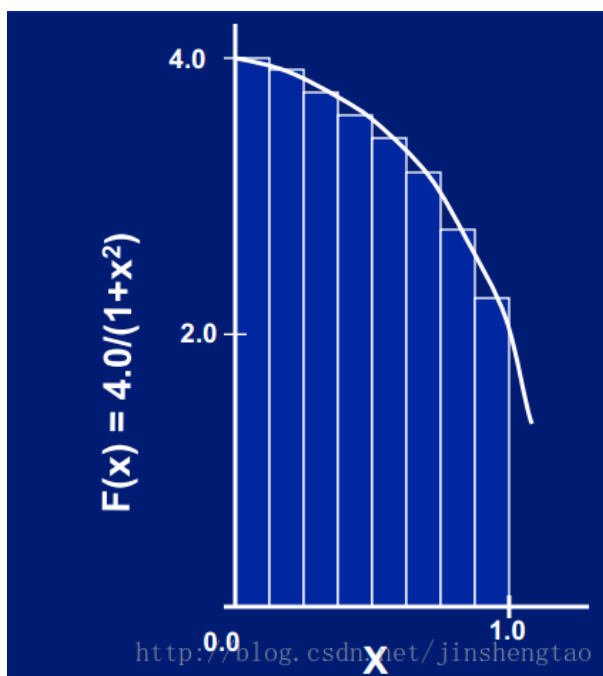
## 2.一则求PI的例子

我们不搞hello world这种入门例子了，直接通过对求PI程序的三种优化方法，来感悟下openmp的魅力

$$\int_0^1 \frac{4}{1+x^2} dx = \pi$$

计算机没法求解连续问题，就用数学插值法来离散化求解

$$\sum_{i=0}^N F(x_i) \Delta x \approx \pi$$



原来的串行代码：

```
[cpp]
1. #include <stdio.h>
2. #include <omp.h>
3. static long num_steps = 100000000;
4. double step;
5. int main ()
6. {
7.     int i;
8.     double x, pi, sum = 0.0;
9.     double start_time, run_time;
10.
11.     step = 1.0/(double) num_steps;
12.
13.
14.     start_time = omp_get_wtime();
15.
16.     for (i=1;i<= num_steps; i++){
17.         x = (i-0.5)*step;
18.         sum = sum + 4.0/(1.0+x*x);
19.     }
20.
21.     pi = step * sum;
22.     run_time = omp_get_wtime() - start_time;
```

SPMD(singleprogram multiple data)优化, 每个线程做各自的统计, 最后通过atomic同步机制汇总

```
[cpp]
1. #include <stdio.h>
2. #include <omp.h>
3.
4. #define MAX_THREADS 4
5.
6. static long num_steps = 100000000;
7. double step;
8. int main ()
9. {
10.     int i,j;
11.     double pi, full_sum = 0.0;
12.     double start_time, run_time;
13.     double sum[MAX_THREADS];
14.
15.     step = 1.0/(double) num_steps;
16.
17.
18.     for(j=1;j<=MAX_THREADS ;j++){
19.         omp_set_num_threads(j);
20.         full_sum = 0.0;
21.         start_time = omp_get_wtime();
22.         #pragma omp parallel private(i)
23.         {
24.             int id = omp_get_thread_num();
25.             int numthreads = omp_get_num_threads();
26.             double x;
27.
28.             double partial_sum = 0;
29.
30.             #pragma omp single
31.             printf(" num_threads = %d",numthreads);
32.
33.             for (i=id;i< num_steps; i+=numthreads){
34.                 x = (i+0.5)*step;
35.                 partial_sum += + 4.0/(1.0+x*x);
36.             }
37.             #pragma omp critical
38.                 full_sum += partial_sum;
39.         }
40.
41.         pi = step * full_sum;
42.         run_time = omp_get_wtime() - start_time;
43.         printf("\n pi is %f in %f seconds %d threds \n ",pi,run_time,j);
44.     }
45. }
```

openMP Loop Parallelism 优化, 注意openmp for loop 语法, 另外注意reduction归约操作

```
[cpp]
1. #include <stdio.h>
2. #include <omp.h>
3. static long num_steps = 100000000;
4. double step;
5. int main ()
6. {
7.     int i;
8.     double x, pi, sum = 0.0;
9.     double start_time, run_time;
10.
11.     step = 1.0/(double) num_steps;
12.     for (i=1;i<=4;i++){
13.         sum = 0.0;
14.         omp_set_num_threads(i);
15.         start_time = omp_get_wtime();
16.         #pragma omp parallel
17.         {
18.             #pragma omp single
19.             printf(" num_threads = %d",omp_get_num_threads());
20.
21.             #pragma omp for reduction(+:sum)
22.             for (i=1;i<= num_steps; i++){
```

```

26.     }
27.     pi = step * sum;
28.     run_time = omp_get_wtime() - start_time;
29.     printf("\n pi is %f in %f seconds and %d threads\n",pi,run_time,i);
30. }
31. }

```

分治法(Divide and Conquer Pattern)优化, openmp task是高级特性, 从openmp3.1后开始支持

## Program: OpenMP tasks (divide and conquer pattern)

```

#include <omp.h>
static long num_steps = 100000000;
#define MIN_BLK 10000000
double pi_comp(int Nstart,int Nfinish,double step)
{
    int i,iblk;
    double x, sum = 0.0,sum1, sum2;
    if (Nfinish-Nstart < MIN_BLK){
        for (i=Nstart;i< Nfinish; i++){
            x = (i+0.5)*step;
            sum = sum + 4.0/(1.0+x*x);
        }
    }
    else{
        iblk = Nfinish-Nstart;
        #pragma omp task shared(sum1)
        sum1 = pi_comp(Nstart, Nfinish-iblk/2,step);
        #pragma omp task shared(sum2)
        sum2 = pi_comp(Nfinish-iblk/2, Nfinish, step);
        #pragma omp taskwait
        sum = sum1 + sum2;
    }
    return sum;
}

int main ()
{
    int i;
    double step, pi, sum;
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        #pragma omp single
        sum = pi_comp(0,num_steps,step);
    }
    pi = step * sum;
}

```

<http://blog.csdn.net/jinshengtao>

### 3.其他例子

生产者与消费者问题优化

原始的串行代码:

```

[cpp]
1.  #include <omp.h>
2.  #ifdef APPLE
3.  #include <stdlib.h>
4.  #else
5.  #include <malloc.h>
6.  #endif
7.  #include <stdio.h>
8.
9.  #define N      10000
10.
11. /* Some random number constants from numerical recipies */
12. #define SEED      2531
13. #define RAND_MULT 1366
14. #define RAND_ADD  150889
15. #define RAND_MOD  714025
16. int randy = SEED;
17.
18. /* function to fill an array with random numbers */
19. void fill_rand(int length, double *a)
20. {
21.     int i;
22.     for (i=0;i<length;i++) {
23.         randy = (RAND_MULT * randy + RAND_ADD) % RAND_MOD;
24.         *(a+i) = ((double) randy)/((double) RAND_MOD);
25.     }
26. }

```

```

30. {
31.     int i;    double sum = 0.0;
32.     for (i=0;i<length;i++)    sum += *(a+i);
33.     return sum;
34. }
35.
36. int main()
37. {
38.     double *A, sum, runtime;
39.     int flag = 0;
40.
41.     A = (double *)malloc(N*sizeof(double));
42.
43.     runtime = omp_get_wtime();
44.
45.     fill_rand(N, A);          // Producer: fill an array of data
46.
47.     sum = Sum_array(N, A);    // Consumer: sum the array
48.
49.     runtime = omp_get_wtime() - runtime;
50.
51.     printf(" In %f seconds, The sum is %f \n",runtime,sum);
52. }

```

并行优化代码，用标记符号通知消费者生产者是否完成

```

[cpp]
1. #include "omp.h"
2. #ifndef APPLE
3. #include <malloc.h>
4. #endif
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. #define N          10000
9. #define Nthreads 2
10.
11. /* Some random number constants from numerical recipies */
12. #define SEED        2531
13. #define RAND_MULT   1366
14. #define RAND_ADD    150889
15. #define RAND_MOD    714025
16. int randy = SEED;
17.
18. /* function to fill an array with random numbers */
19. void fill_rand(int length, double *a)
20. {
21.     int i;
22.     for (i=0;i<length;i++) {
23.         randy = (RAND_MULT * randy + RAND_ADD) % RAND_MOD;
24.         *(a+i) = ((double) randy)/((double) RAND_MOD);
25.     }
26. }
27.
28. /* function to sum the elements of an array */
29. double Sum_array(int length, double *a)
30. {
31.     int i;    double sum = 0.0;
32.     for (i=0;i<length;i++)    sum += *(a+i);
33.     return sum;
34. }
35.
36. int main()
37. {
38.     double *A, sum, runtime;
39.     int numthreads, flag = 0, flg_tmp=0;
40.
41.     omp_set_num_threads(Nthreads);
42.
43.     A = (double *)malloc(N*sizeof(double));
44.
45.     #pragma omp parallel
46.     {
47.         #pragma omp master
48.         {
49.             numthreads = omp_get_num_threads();

```

```
53.         exit(-1);
54.     }
55.     runtime = omp_get_wtime();
56. }
57. #pragma omp barrier
58.
59. #pragma omp sections
60. {
61.     #pragma omp section
62.     {
63.         fill_rand(N, A);
64.         #pragma omp flush
65.         flag = 1;
66.         #pragma omp flush (flag)
67.     }
68.     #pragma omp section
69.     {
70.         while (1){
71.             #pragma omp flush (flag)
72.             #pragma omp atomic read
73.             flg_tmp = flag;
74.             if(flg_tmp == 1) break;
75.         }
76.
77.         #pragma omp flush
78.         sum = Sum_array(N, A);
79.     }
80. }
81. #pragma omp master
82.     runtime = omp_get_wtime() - runtime;
83. }
84.
85. printf(" with %d threads and %lf seconds, The sum is %lf \n", numthreads, runtime, sum);
86. }
```

链表遍历、斐波那契递归问题的优化：

原始串行代码：

```
[cpp]
1. #include <stdlib.h>
2. #include <stdio.h>
3. #include <omp.h>
4.
5. #ifndef N
6. #define N 5
7. #endif
8. #ifndef FS
9. #define FS 38
10. #endif
11.
12. struct node {
13.     int data;
14.     int fibdata;
15.     struct node* next;
16. };
17.
18. int fib(int n) {
19.     int x, y;
20.     if (n < 2) {
21.         return (n);
22.     } else {
23.         x = fib(n - 1);
24.         y = fib(n - 2);
25.         return (x + y);
26.     }
27. }
28.
29. void processwork(struct node* p)
30. {
31.     int n;
32.     n = p->data;
33.     p->fibdata = fib(n);
34. }
```

```
38.     struct node* head = NULL;
39.     struct node* temp = NULL;
40.
41.     head = malloc(sizeof(struct node));
42.     p = head;
43.     p->data = FS;
44.     p->fibdata = 0;
45.     for (i=0; i< N; i++) {
46.         temp = malloc(sizeof(struct node));
47.         p->next = temp;
48.         p = temp;
49.         p->data = FS + i + 1;
50.         p->fibdata = i+1;
51.     }
52.     p->next = NULL;
53.     return head;
54. }
55.
56. int main(int argc, char *argv[]) {
57.     double start, end;
58.     struct node *p=NULL;
59.     struct node *temp=NULL;
60.     struct node *head=NULL;
61.
62.     printf("Process linked list\n");
63.     printf(" Each linked list node will be processed by function 'processwork()'\n");
64.     printf(" Each ll node will compute %d fibonacci numbers beginning with %d\n",N,FS);
65.
66.     p = init_list(p);
67.     head = p;
68.
69.     start = omp_get_wtime();
70.     {
71.         while (p != NULL) {
72.             processwork(p);
73.             p = p->next;
74.         }
75.     }
76.
77.     end = omp_get_wtime();
78.     p = head;
79.     while (p != NULL) {
80.         printf("%d : %d\n",p->data, p->fibdata);
81.         temp = p->next;
82.         free (p);
83.         p = temp;
84.     }
85.     free (p);
86.
87.     printf("Compute Time: %f seconds\n", end - start);
88.
89.     return 0;
90. }
```

## 并行优化代码

看了openmp论坛，递归那边前20个不用并行优化，否则性能反而降低了

### [cpp]

```
1. #include <omp.h>
2. #include <stdlib.h>
3. #include <stdio.h>
4.
5.
6. #ifndef N
7. #define N 5
8. #endif
9. #ifndef FS
10. #define FS 38
11. #endif
12.
13. typedef struct node {
14.     int data;
15.     int fibdata;
16.     struct node* next;
17. }
```

```

20. void processwork(node* p);
21. int fib(int n);
22.
23. int fib(int n)
24. {
25.     int x, y;
26.     if (n < 2) {
27.         return (n);
28.     } else {
29.         if (n < 20)
30.             return fib(n-1)+fib(n-2);
31. #pragma omp task shared(x)
32.     x = fib(n - 1);
33. #pragma omp task shared(y)
34.     y = fib(n - 2);
35. #pragma omp taskwait
36.     return (x + y);
37.     }
38. }
39.
40. void processwork(node* p)
41. {
42.     int n, temp;
43.     n = p->data;
44.     temp = fib(n);
45.
46.     p->fibdata = temp;
47.
48. }
49.
50. node* init_list(node* p)
51. {
52.     int i;
53.     node* head = NULL;
54.     node* temp = NULL;
55.
56.     head = (node*)malloc(sizeof(node));
57.     p = head;
58.     p->data = FS;
59.     p->fibdata = 0;
60.     for (i=0; i < N; i++) {
61.         temp = (node*)malloc(sizeof(node));
62.         p->next = temp;
63.         p = temp;
64.         p->data = FS + i + 1;
65.         p->fibdata = i+1;
66.     }
67.     p->next = NULL;
68.     return head;
69. }
70.
71. int main()
72. {
73.     double start, end;
74.     struct node *p=NULL;
75.     struct node *temp=NULL;
76.     struct node *head=NULL;
77.
78.     printf("Process linked list\n");
79.     printf("  Each linked list node will be processed by function 'processwork()'\n");
80.     printf("  Each ll node will compute %d fibonacci numbers beginning with %d\n", N, FS);
81.
82.     p = init_list(p);
83.     head = p;
84.
85.     start = omp_get_wtime();
86.
87.     #pragma omp parallel
88.     {
89.         #pragma omp master
90.         printf("Threads:      %d\n", omp_get_num_threads());
91.
92.         #pragma omp single
93.         {
94.             p=head;
95.             while (p) {
96.                 #pragma omp task firstprivate(p) //first private is required
97.                 {

```



```
101.         }
102.     }
103. }
104.
105.     end = omp_get_wtime();
106.     p = head;
107.     while (p != NULL) {
108.         printf("%d : %d\n",p->data, p->fibdata);
109.         temp = p->next;
110.         free (p);
111.         p = temp;
112.     }
113.     free (p);
114.
115.     printf("Compute Time: %f seconds\n", end - start);
116.
117.     return 0;
118. }
```

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/jinshengtao/article/details/79120067>

文章标签：openMP

【正在直播】为什么80%的程序员，这次都站全栈工程师？

随着IT市场需求的变化，全栈工程师似乎已成为未来发展趋势。很多Flag公司都已经声称只招Full Stack的员工，那么为什么全栈工程师最受欢迎？一个案例带你先睹为快！

[查看更多>>](#)

18527

[查看更多>>](#)

想对作者说点什么？

[我来说一句](#)

【并行计算】基于OpenMP的并行编程（#pragma omp parallel for）

百度云盘：MPI 并行计算教材 <https://pan.baidu.com/s/1htgGZh6> 密码：bt1p 我们目前的计算机都是基于冯诺伊曼结构的，在MIMD作为主要研究对象的系统中，分为...

 eric\_e 2018-01-24 22:46:06 阅读数：203

关于利用Openmp中使用的函数

Openmp是一项并行化技术，是可以提高串行化程序的运行效率的，但需要使用正确的时间函数来进行衡量。首先，先提出unix/linux下的内核时间获取函数 1.clock()函数 先看其在MSDN中的...

 sinat\_15799399 2015-05-13 20:14:00 阅读数：2175


openmp在多重循环内的简单使用及其详解

由于项目需求，在三重循环内加入了并行计算，但由于只能在内层循环加入，而内层循环只有32维度，因此速度提高的也就那么几毫秒。在此 不再将代码贴出！ 以下是转载的别人博客中的详细讲解，很不错！ ...

 Allyli0022 2016-09-29 15:44:40 阅读数：5607

OpenMP学习

<http://openmp.org/wp/> 传统的单线程编程方式难以发挥多核CPU的强大功能，于是多核编程应运而生。多核编程可以认为是对多核环境下编程做了一些多线程抽象，提供一些简单的API，使得用...

 CHS007chs 2014-12-10 16:50:23 阅读数：1122

OpenMP for Android初学记录

OpenMP是一种应用程序接口(API)，支持多平台共享库的C/C++/Fortran多处理器编程，可以运行在绝大多数处理器架构和操作系统上，包括Solaris