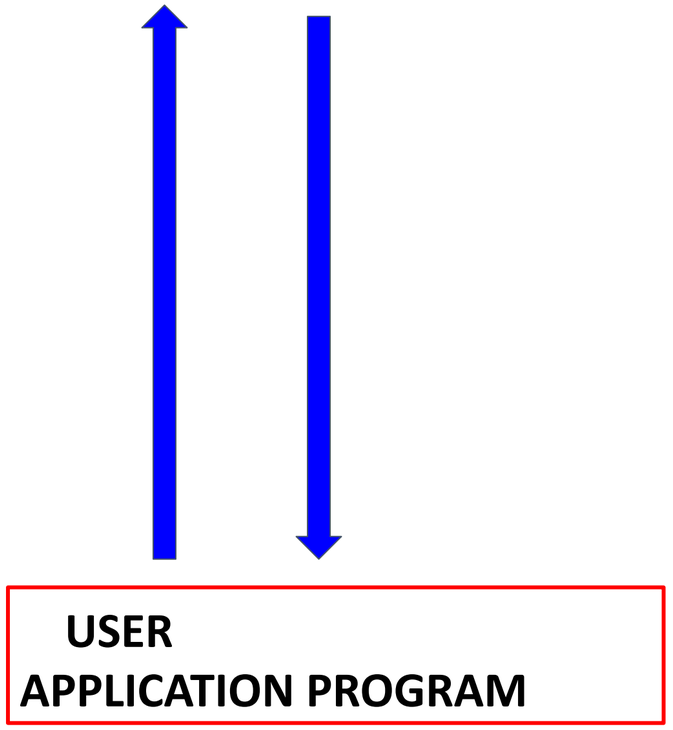
Using profiling tools like **gprof**

- Parallel programs performs well when atomic and synchronization constructs are reduced.
  - total removal not possible in all cases.
- Locality of reference importance for matrix operation
  - Spatial and Temporal Locality
- Access latency
  - In the increasing order: L1 cache, L2 cache, L3 cache, Main Memory, Hardisk (secondary storage)
- Sequential Consistency
  - Output of parallel program should be same as one sequential execution
  - Inter iteration Dependence (RAW/WAR/WAW) should not be there.
-

Operating System Structure

- Hardware at bottom
  - keyboard, mouse, network card etc.
- Operating system Kernel on top of it
  - Linux kernel
- Application Level Interface
  - Unix command line
- GUI interface
  - GNome

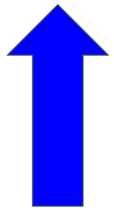


Todays Lecture: Profiling Parallel Programs

- Program
  - Need Space, Time, and Energy for computation
- Time - bash command
- gettimeofday(), and clock() functions.
- gprof
  - Performance Analysis Tool
  - Extended version of the tool prof
  - gprof can do callgraph collecting and printing

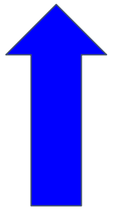
Parallel Matrix Addition

```
#pragma omp parallel for
num_threads(8)
        for(int i=0;i<size;i++)
            for(int j=0;j<size;j++)
                C[i][j]=A[i][j]+B[i][j];
```



Row Major Access

```
#pragma omp parallel for
num_threads(8)
        for(int i=0;i<size;i++)
            for(int j=0;j<size;j++)
                C[j][i]=A[j][i]+B[j][i];
```



Column Major Access

Sample Time Measurement Using time Command

```
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ gcc matrixadd_row.c -fopenmp -o row
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ time ./row 3

real    0m1.629s
user    0m2.067s
sys     0m0.337s
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ gcc matrixadd_column.c -fopenmp -o col
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ time ./col 3

real    0m4.275s
user    0m21.490s
sys     0m0.565s
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$
```

Time Command Output: Meanings

- Real
  - Time from start to finish of the execution.
- User
  - Amount of CPU time spend in user mode.
- Sys
  - Amount of time spend in kernel mode.

Sample time measurement using time command

```
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ gcc matrixadd_row.c -fopenmp -o row
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ time ./row 3

real    0m1.629s
user    0m2.067s
sys     0m0.337s
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ gcc matrixadd_column.c -fopenmp -o col
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ time ./col 3

real    0m4.275s
user    0m21.490s
sys     0m0.565s
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$
```

User time is total time taken by 8 threads.
The program was run using 8 threads.
Time per one core is around 1/8th.

Serial and Parallel Execution: Row Major addition

```
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ gcc matrixadd_row.c -o   row
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ gcc matrixadd_row.c -o   rowserial
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ gcc matrixadd_row.c -fopenmp -o   rowparallel
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ time ./rowserial 3

TIME =373.191118

real    0m2.008s
user    0m1.747s
sys     0m0.260s
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ time ./rowparallel 3

TIME =126.043081

real    0m1.660s
user    0m2.178s
sys     0m0.352s
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$
```

TIME value is in Milliseconnds (Ms)
373 Ms = 0.373 Ms **(Serial)**
126 Ms = .126 Ms (0.126 x 8 = 1.008) **(Parallel)**
TIME measured only for matrix addition loop shown in earlier slide.
Not for entire program.

Serial and Parallel Execution: Column Major addition

```
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ gcc matrixadd_column.c -fopenmp  -o   colparallel
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ gcc matrixadd_column.c  -o   colserial
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ time ./colparallel 3

TIME =2677.949905

real    0m4.271s
user    0m21.531s
sys     0m0.555s
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$ time ./colserial 3

TIME =2193.084002

real    0m3.862s
user    0m3.511s
sys     0m0.348s
unnikrishnan@unnikrishnan-X510UNR:~/matrixadd$
```

TIME value is in Milliseconnds (Ms)
2668 Ms = (2668x8= 21344= 21.44 Seconds) (**Parallel**)
2193 Ms **(Serial)**
Time for Matrix addition loop shown in earlier slide.
Not for Entire program

times() function

- Signature : clock_t times(struct tms *buf);
- Returns the number of clock ticks elapsed.
- Stores the current process times in buf
- struct tms {
  - clock_t tms_utime; // user time of process
  - clock_t tms_stime; //system time spend in kernel

  ....

  }

computing time using gettimeofday() function

- Function Signature
  - int gettimeofday(struct timeval *, struct timezone *);
- struct timeval
  - stores number of seconds (tv_sec) and microseconds(tv_usec)

    since last Epoch (1970-01-01 00:00:00 +0000 -UTC)
- This can be used to get current relative time.
- To find time elapsed during execution of a code block
  - Get time immediately before entering the code block (t1)
  - Get time immediately after finishing execution of the code block (t2)
  - Value (t2 - t1) will give time elapsed during the execution.
  -

Value of “TIME” (see previous slide)

```
double rtclock()
{
   struct timezone Tzp;
   struct timeval Tp;
   int stat;
   stat = gettimeofday (&Tp, &Tzp);
   if (stat != 0) printf("Error”);
   return(Tp.tv_sec + Tp.tv_usec*1.0e-6);
}
```

```
double t1= rtclock();
  ….. /* Do Computation, matrix-add
*/
double t2= rtclock();
double t3= (t2-t1);
printf(“TIME=%f\n”, t3);
```

clock() function

- Signature - clock_t clock(void);
- Approximate processor time used by the program.

```
#include <time.h>

clock_t t1, t2;
double cpu_time;

t1 = clock()
... /* Do computation */
 t2 = clock();
cpu_time= ((double) (t2 - t1)) / CLOCKS_PER_SEC; // Time in Seconds

CLOCKS_PER_SEC is having a value  1000000
```

GPROF - Usage

- Compile program with -pg option
  - gcc -pg program.c -o prog
- Run the program
  - ./prog [arguments]
  - generates gmon.out
- Run gprof with options
  - gprof [options] ./prog
  - it prints profiling information

Sample program

```
int fun1(){/*prg1.c begin */
    printf("Inside fun1()\n");
    for(int i=0;i<20000;i++);
    return 1 ;
}/*prg1.c end */
int  fun2() {/* prg.c begin */
    printf("Inside fun2\n");
        for(int i=0;i<100000000;i++);
    fun1();
    return 1;
}
```

```
static void fun3(){
    printf("Inside fun3()\n");
    for(int i=0;i<40000;i++);
    return;
}
int main() {
    printf("\n Inside main()");
    for(int i=0;i<100000000;i++);
  fun2();
    fun3();
    return 0;
```

Running the code for profile information

- Compile with profiling option
  - gcc -pg prg.c prg1.c -o gprof_test
- Run the program
  - ./gprof_test
  - a file with name gmon.out will be created.
- Run with gprof tool
  - gprof gprof_test gmon.out > output.txt
  - > redirects output from screen to the file output.txt.

Flat profile

```
unnikrishnan@unnikrishnan-X510UNR: ~/gprof
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ ls
prg1.c  prg.c
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ gcc -pg prg.c prg1.c -o gprof_test
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ ./gprof_test
Inside main()
Inside fun2
Inside fun1()
Inside fun3()
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ ls
gmon.out  gprof_test  prg1.c  prg.c
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ gprof ./gprof_test gmon.out > output.txt
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ wc -l output.txt
157 output.txt
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ head -n 12 output.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 48.15      0.57     0.57        1   572.98   572.98  fun3
 25.34      0.87     0.30        1   301.57   301.57  fun1
 14.36      1.05     0.17                             main
 12.67      1.20     0.15        1   150.79   452.36  fun2

 %         the percentage of the total running time of the
time       program used by this function.
unnikrishnan@unnikrishnan-X510UNR:~/gprof$
```

command head with option -n 12 prints first 12 lines of the file

total number of lines in output.txt is 157

Fields in the flat profile

- index - Unique value for each element in the table
- %time - Percentage of total time spent by the function
- Cumulative seconds
  - Total time of this function and those listed above in Seconds
- Self seconds
  - Number of seconds by this function alone
- calls
  - Number of times the function is called, if profiled. else blank
- Self/per call and total/per call -average value of all calls.
- name- Name of the current function

Flat profile

```
unnikrishnan@unnikrishnan-X510UNR: ~/gprof
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ ls
prg1.c  prg.c
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ gcc -pg prg.c prg1.c -o gprof_test
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ ./gprof_test
Inside main()
Inside fun2
Inside fun1()
Inside fun3()
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ ls
gmon.out  gprof_test  prg1.c  prg.c
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ gprof ./gprof_test gmon.out > output.txt
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ wc -l output.txt
157 output.txt
unnikrishnan@unnikrishnan-X510UNR:~/gprof$ head -n 12 output.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 48.15      0.57     0.57        1   572.98   572.98  fun3
 25.34      0.87     0.30        1   301.57   301.57  fun1
 14.36      1.05     0.17                             main
 12.67      1.20     0.15        1   150.79   452.36  fun2

 %         the percentage of the total running time of the
time       program used by this function.
unnikrishnan@unnikrishnan-X510UNR:~/gprof$
```

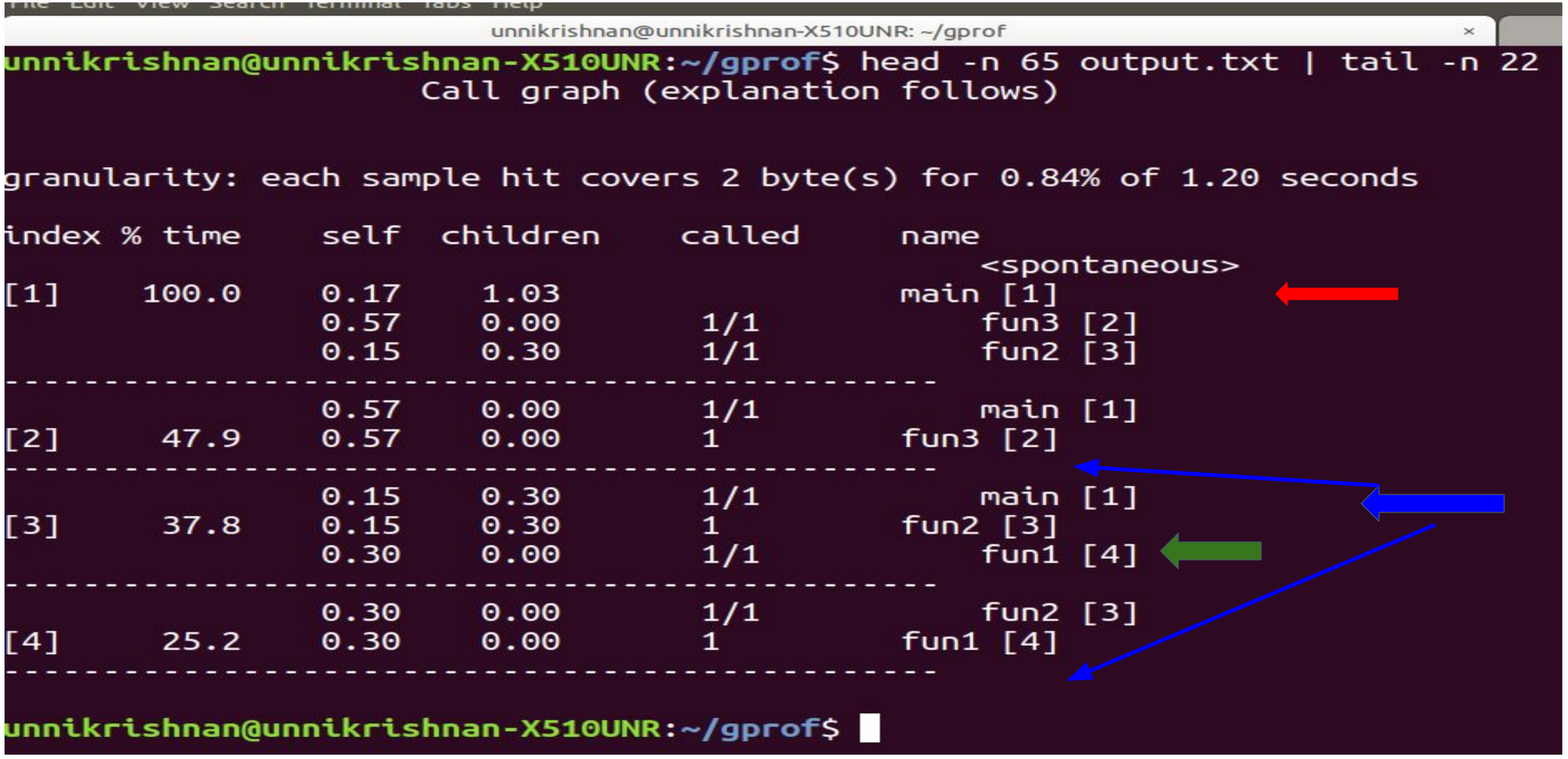
command head with option -n 12 prints first 12 lines of the file

total number of lines in output.txt is 157

Call graph profile

- index - unique id for each entry in the table
- %time
  - Percentage of total time spent by the function and its children
- Self seconds
  - Number of seconds by this function alone
- children
  - total amount of time propagated into this function by its children
- called - Number of times this function is called.
- name- name of the current function.
- <spontaneous>
  - if parent of function cannot be determined, this is printed for parent name.

Call Graph - Profile information



gprof - reducing output text

- gprof -b ./gprof_test gmon.out > output.txt
  - removes the verbose information.
  - output was just 36 lines compared 157 without -b option.
- gprof -p -b ./gprof_test gmon.out > output.txt
  - prints only flat profile
  - output was just nine lines
- gprof -pfun1 -b ./gprof_test gmon.out > output.txt
  - prints flat profile only for fun1
- gprof -P -b ./gprof_test gmon.out > output.txt /* P - capital letter */
  - removes flat profile
- gprof -q -b ./gprof_test gmon.out > output.txt
  - prints only call graph profile.
- gprof -qfun1 -b ./gprof_test gmon.out > output.txt /*-Q to remove call graph profile */
  - prints call graph profile only for fun1

Sparse Matrix

Matrix with four rows and ten columns.

Non zero elements are twelve.

0	0	0	15	0	5	0	0	0	0
0	0	0	5	8	0	0	0	10	12
0	20	0	0	12	0	0	13	10	0
0	0	43	0	0	0	0	0	0	12

0	2	6	10	12

3	15	5	5	3	5	4	8	8	10	9	12
1	20	4	12	7	13	8	10	2	43	9	12

Total Space in initial array - (4x10) elements

Modified array - (24+5)=29