

Parallel programming

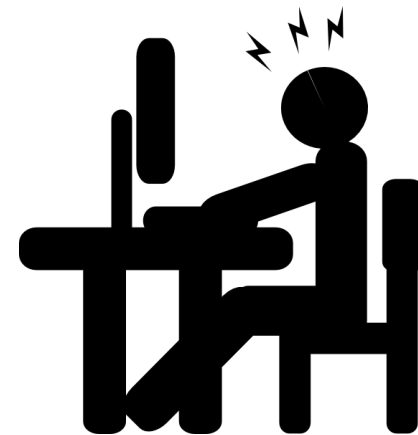
Introduction





What is the aim of the labs?

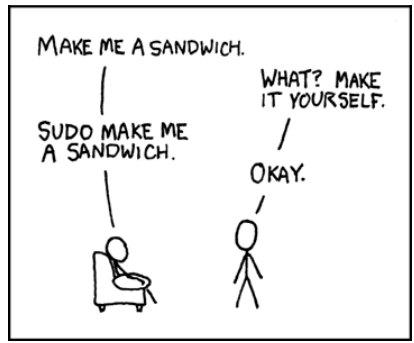
- To get the feel for parallel programming
 - 1) Understand what makes the parallelisation **complicated**
 - 2) Which **problems** can occur during the parallelisation
 - 3) What can be a **bottleneck**
 - 4) How to think about **algorithms** from the parallelisation point of view
- To get basic skills in common parallel programming frameworks
 - 1) for Multicore processors – C++11 threads, OpenMP
 - 2) for Computer clusters – OpenMPI



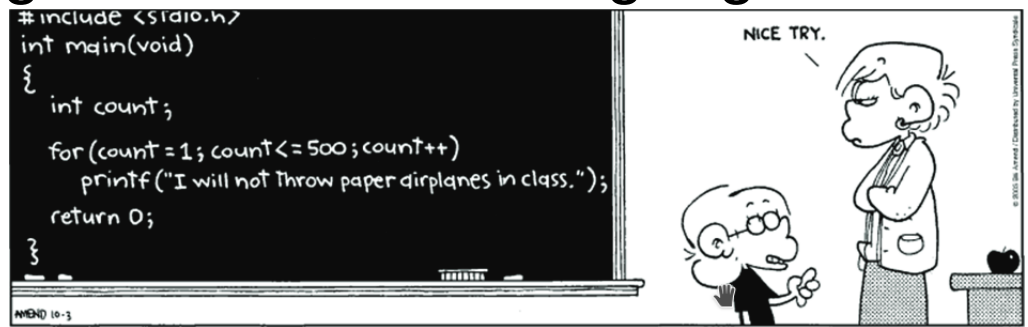


What this course requires?

- Basic skills with Linux – shell, ssh, etc.



- Knowledge of C and C++ language



- Analytical thinking and opened mind





Web

- Course page
<https://cw.fel.cvut.cz/wiki/courses/b4m35pag/start>
 - Plan of the labs, grading



Parallel programming – the first cut

No questions?

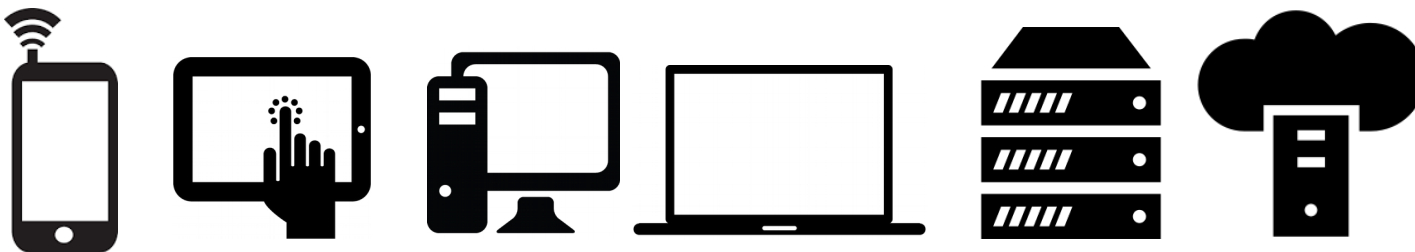
Let's start with our business!





Why should you care about it?

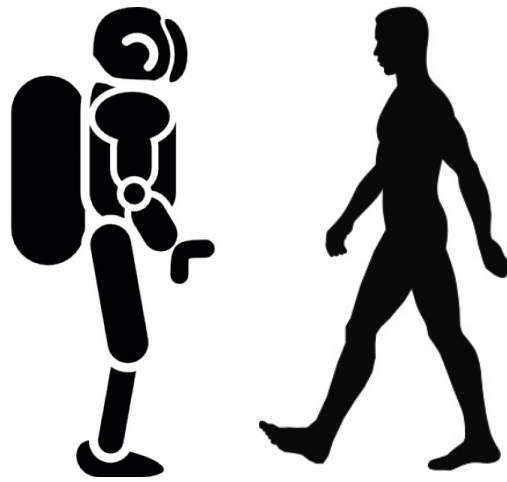
- Parallel computing is a dominant player in scientific and cluster computing. Why?
 - Moore law is reaching its limits
 - Increase in transistor density is limited
 - Memory access time has not been reduced at a rate comparable with processing speed
- How to get out of this trap?
 - Most promising approach is to have multiple cores on a single processor.
 - Parallel computing can be found at many devices today:





Ok; However, It should be task for compiler and not for me!!!

- Yes, compiler can help you, but without your guidance, it is not able pass all the way to the successful result.
 - Parallel programs often look very different than sequential ones.
 - An efficient parallel implementation of a serial program may not be obtained by simply parallelizing each step.
 - Rather, the best parallelization may be obtained by stepping back and devising an entirely new algorithm.

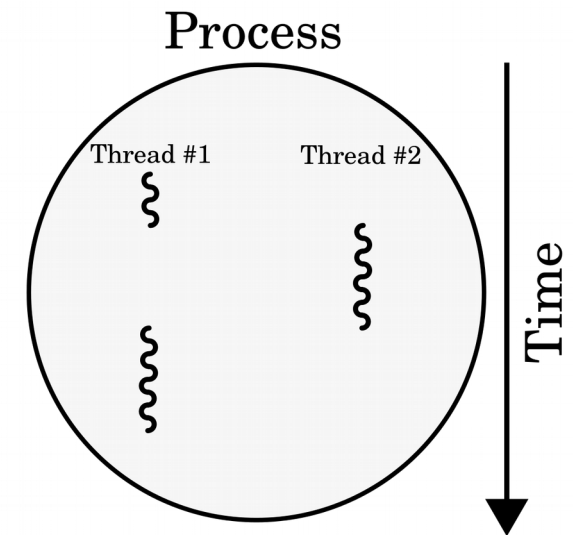


Basics of Parallel programming theory

Terms



- Program
 - Collection of instructions designed to perform a group of coordinated functions
- Process
 - Instance of a program that is being executed.
 - Multiple processes are typically independent
 - It has its own memory space.
- Thread
 - Sequence of instructions that is managed independently by system scheduler.
 - Subset of process
 - Multiple threads within process share the memory space.
- Task
 - Unit of execution

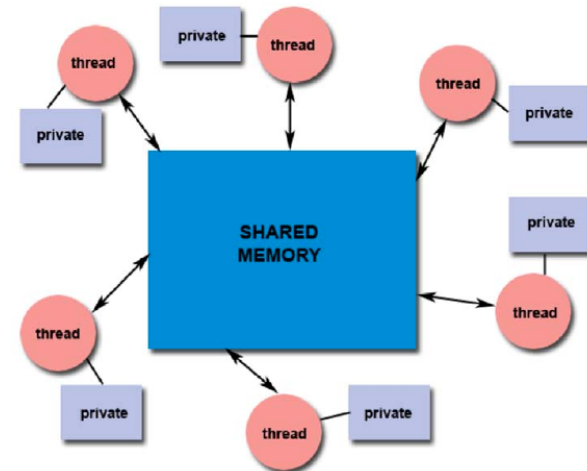


Basics of Parallel programming theory

Memory architectures

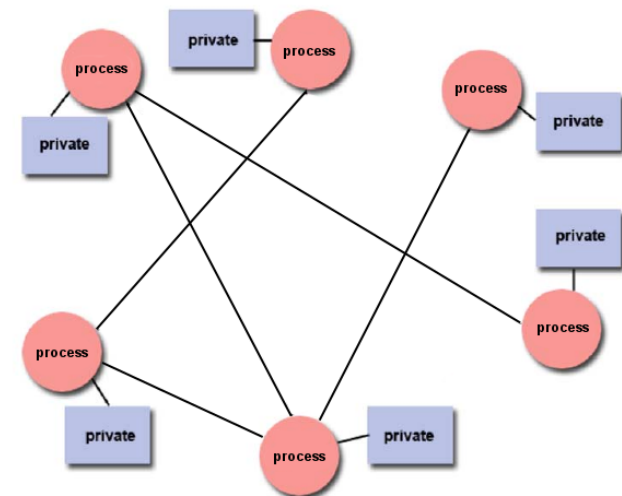
- **Shared memory**

- All functional units share the common memory space.
- When a functional unit share the value in the common memory space another functional unit can access this value.



- **Distributed memory**

- Each functional unit has its own private memory space.
- When two or more units need to share a value, they have to exchange this value by a message transmitted through the network.

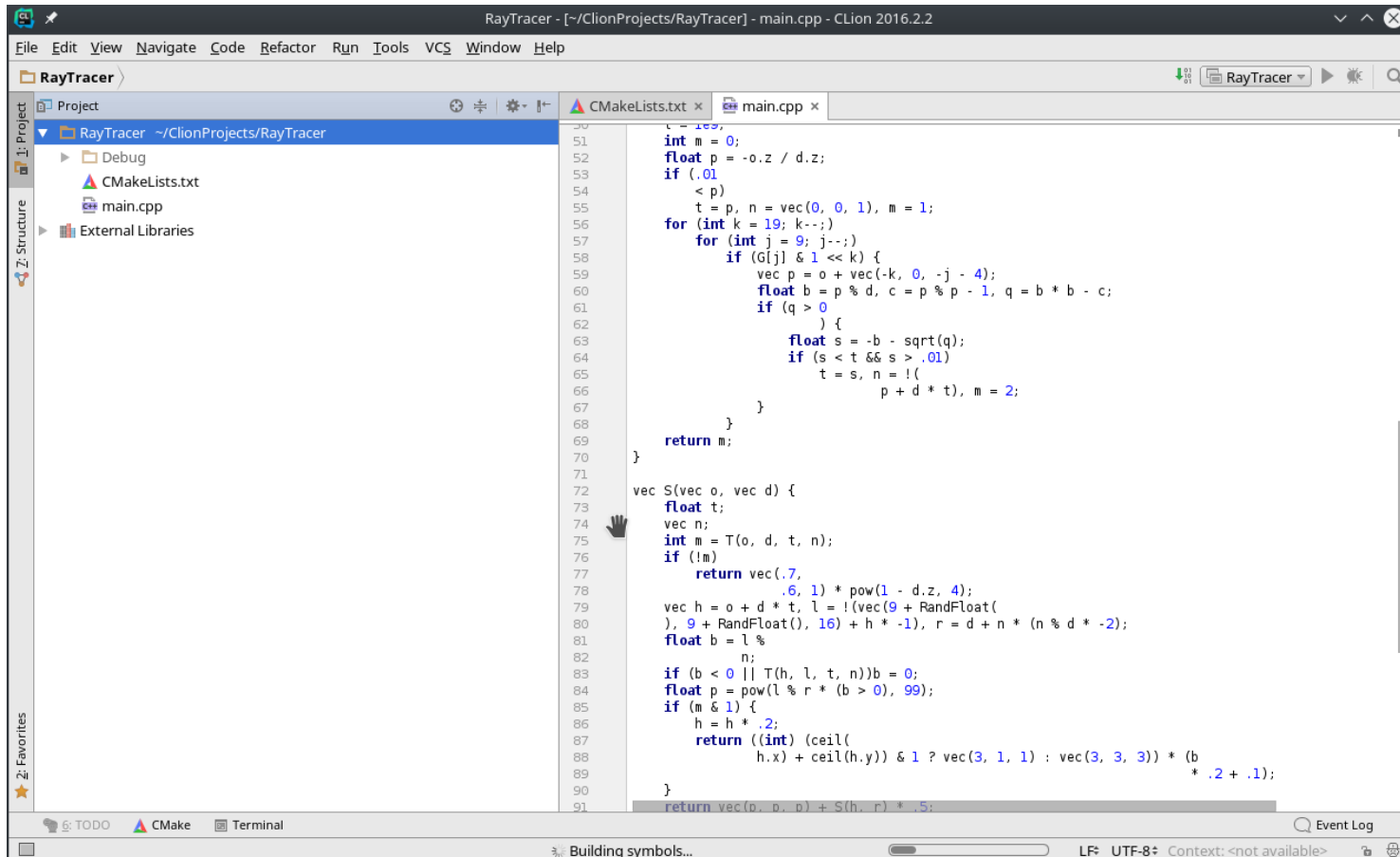




That was theory and now something practical!

Clion IDE

- Licence server - <https://turnkey.felk.cvut.cz/>



```
RayTracer - [~/ClionProjects/RayTracer] - main.cpp - CLion 2016.2.2
File Edit View Navigate Code Refactor Run Tools VCS Window Help
RayTracer
Project
  RayTracer ~/ClionProjects/RayTracer
    Debug
    CMakeLists.txt
    main.cpp
    External Libraries
Structure
Z: Favorites
main.cpp
51 int m = 0;
52 float p = -o.z / d.z;
53 if (.01
54 < p)
55 t = p, n = vec(0, 0, 1), m = 1;
56 for (int k = 19; k--;)
57 for (int j = 9; j--;)
58 if (G[j] & 1 << k) {
59 vec p = o + vec(-k, 0, -j - 4);
60 float b = p % d, c = p % p - 1, q = b * b - c;
61 if (q > 0
62 ) {
63 float s = -b - sqrt(q);
64 if (s < t && s > .01)
65 t = s, n = !(
66 p + d * t), m = 2;
67 }
68 }
69 return m;
70 }
71 }
72 vec S(vec o, vec d) {
73 float t;
74 vec n;
75 int m = T(o, d, t, n);
76 if (!m)
77 return vec(.7,
78 .6, 1) * pow(1 - d.z, 4);
79 vec h = o + d * t, l = !(vec(9 + RandFloat(
80 ), 9 + RandFloat(), 16) + h * -1), r = d + n * (n % d * -2);
81 float b = l %
82 n;
83 if (b < 0 || T(h, l, t, n)) b = 0;
84 float p = pow(l % r * (b > 0), 99);
85 if (m & 1) {
86 h = h * .2;
87 return ((int) (ceil(
88 h.x) + ceil(h.y)) & 1 ? vec(3, 1, 1) : vec(3, 3, 3)) * (b
89 * .2 + .1);
90 }
91 }
92 return vec(p, p, p) + S(h, r) * .5;
93 }
```



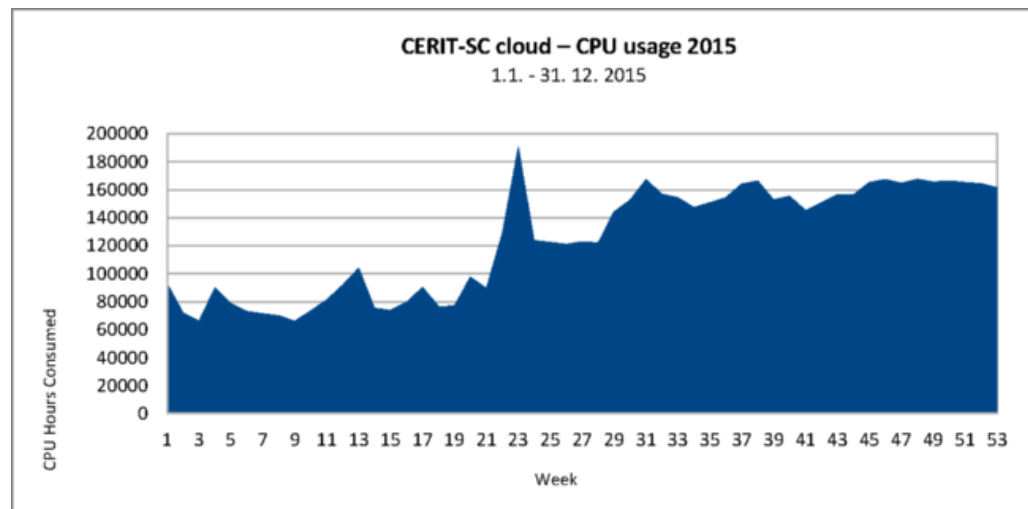
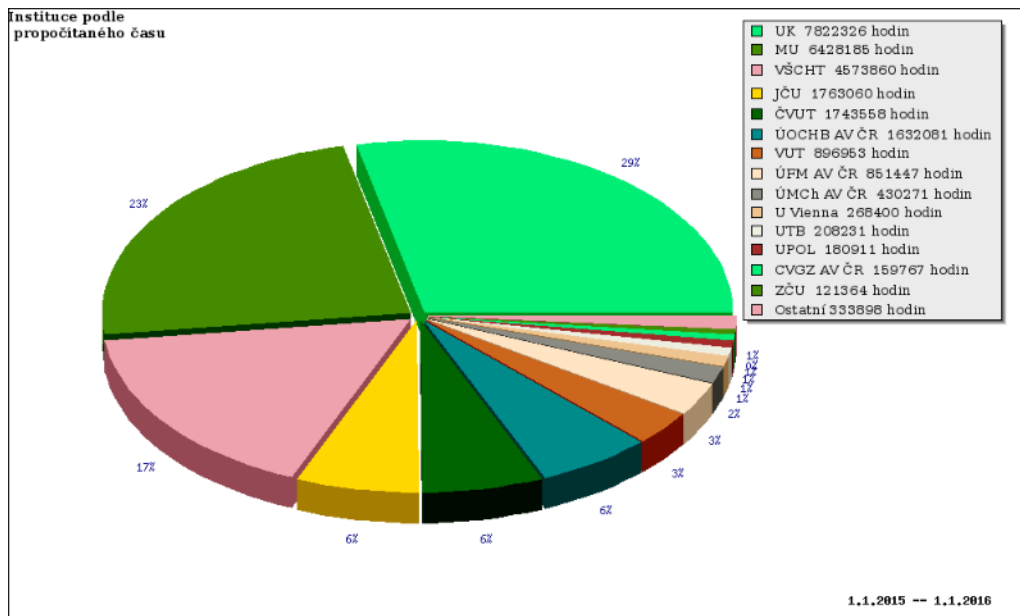
Hello world for free

- Live example and walk-through
 - Create project, Build code, Run code, Debug code,
 - Code profiling – valgrind (tools callgrind, cachegrind)

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "Hello, World!" << std::endl;
5     return 0;
6 }
```

Metacentrum system

- operates and manages **distributed computing** infrastructure consisting of computing and storage resources owned by **CESNET**
- MetaCentrum membership is free for researchers and students of academic institutions in the Czech Republic





MetaCentrum – Sign up

- How to sign up



MetaCentrum



Metacentrum – How to run code?

- Example of the execution of a program
 - qsub
 - module
 - \$SCRATCHDIR
 - \$PSB_O_WORKDIR
- Running jobs in MetaCentrum
 - https://wiki.metacentrum.cz/wiki/Running_jobs_in_scheduler
- Detailed description of the scheduling system
 - https://wiki.metacentrum.cz/wiki/Scheduling_system_-_detailed_description
- Application modules
 - https://wiki.metacentrum.cz/wiki/Application_modules



That was nice, wasn't it?

Thank you for your attention...

