Parallel programming

Introduction

FAKULTA
ELEKTROTECHNICKÁ
ČVUT V PRAZE

Why should you care about it?

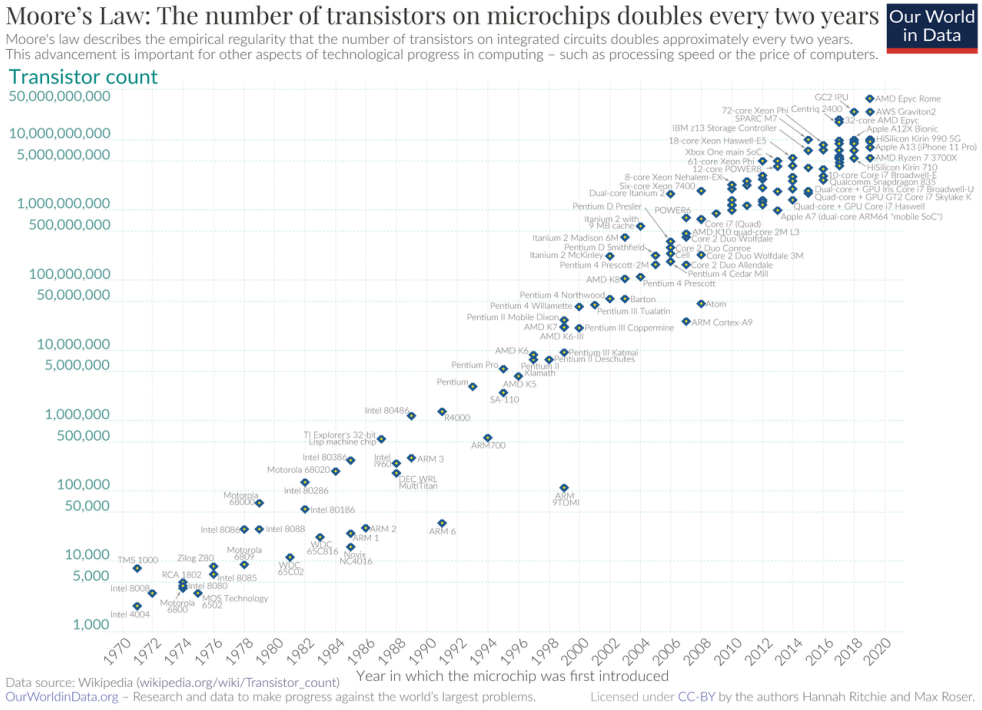
- Sometimes you **want to get the result faster** – the algorithm with big amount of computation / big amount of data

  Application: scientific world (*simulations, calculations*), big data computing (*faster proccessing, databases*), machine learning, deep learning

- Sometime you **have limited time to fulfill task**, sequential way is too slow – real time processing
- Benefit: some general principles are applicable in thinking about architecture of separate programs over related tasks

Why should you care about it?

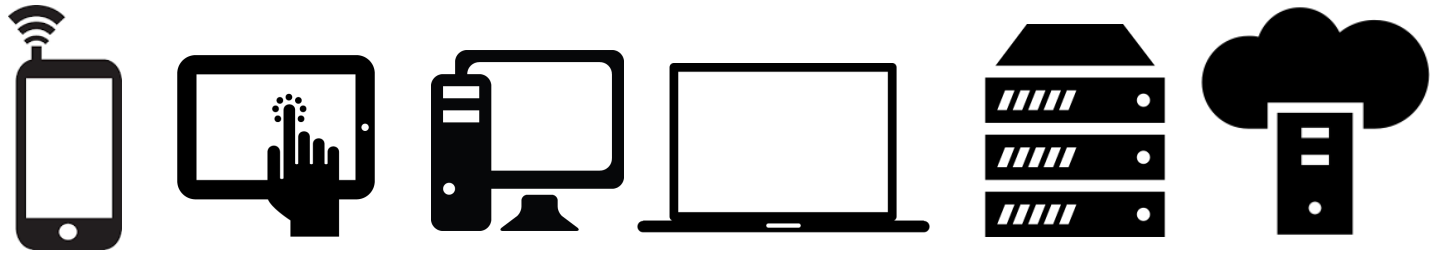
- Parallel computing is a dominant player in scientific and cluster computing. Why?
  - Moore law (number of transistors doubles about every two years; for same price, price per power halving) is reaching its limits
    - Increase in transistor density is limited
    - Memory access time has not been reduced at a rate comparable with processing speed



2. This chart shows the Linley Group's "Cost Per Transistor" curve (2017) taken from Cadence's "Breakfast Bytes" blog.

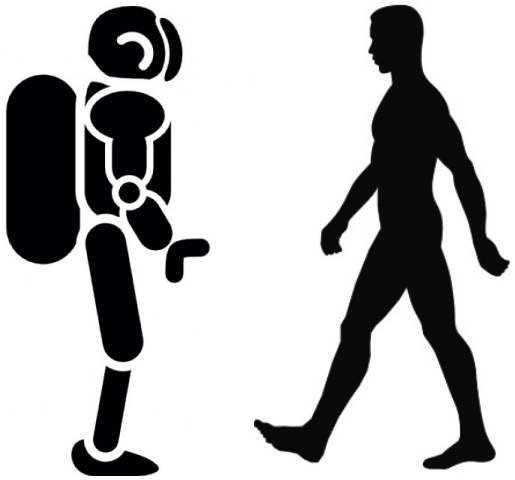
Why should you care about it?

- How to get out of this trap?
  - Most promising approach is to have multiple cores on a single processor
  - Number of cores growing, speed per core growing slower
  - Today's desktop computers 2023 offer
    Intel Core i9-13900KS - 24 cores, 32 threads, 3,2GHz (TDP 253W), Boost 6 GHz.
    AMD Ryzen 9 7950X3D - 16 cores, 32 threads, 4,2GHz (TDP 120W), Boost 5,7 GHz
  - 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
  - Instruction level paralelization

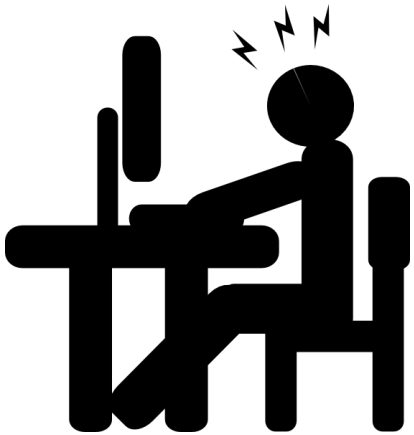


What is the aim of labs?

- To get the feel for parallel programming
  1) Understand what makes the parallelisation **complicated**
  2) Which **problems** can occur during the paralellisation
  3) What can be a **bottleneck**
  4) How to think about **algorithms** from the paralellisation point of view

Familiar terms: race condition, false sharing, synchonizaton, deadlocks, communication overhead, work disbalance, idling, another design of algorithm vs. sequential version

- To get basic skills in common parallel programming frameworks
  1) for Multicore processors
  2) for Computer clusters
  3) for GPU (nice opportunity to play with)



Seminar topics

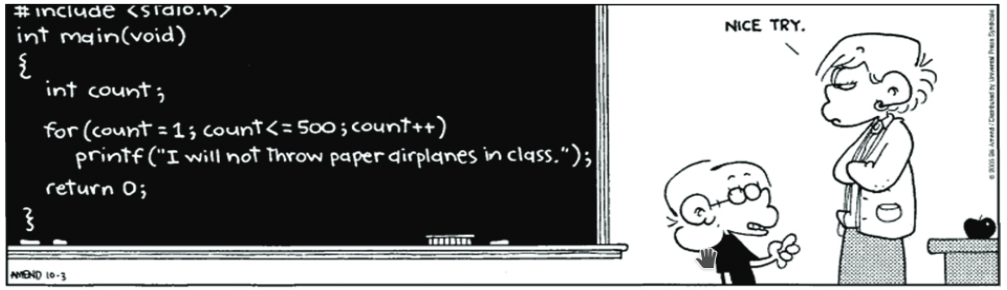
- OpenMP – for Multicore processors, easy way to parallelize originally sequential code, UMA concept
- MPI – for Computer clusters, concept of units comunicating through messages, NUMA concept
- Numba – computation on GPU
- Theoretical seminars – helps to prepare for the exam

Course web

- Course page https://cw.fel.cvut.cz/b231/courses/pag/start
  - Detailed plan of the labs, grading

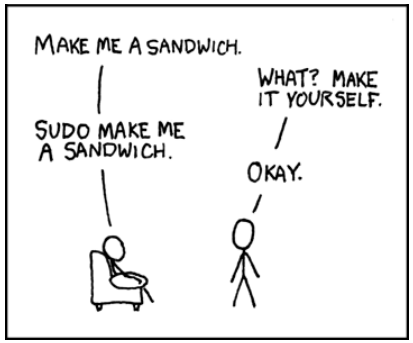
What does this course require?

- Knowledge of C, C++, basics of Python



- Analytical thinking and being open-minded

- Basic skills with Linux – shell, ssh, etc. (for MetaCentrum)



Setting up

- Installation at home
- Be prepared for coding next week
- Small helloworld examples prepared for you to check if enviroment runs smoothly
- Recommendations follow

Our recommendations

Linux, Mac OS, Windows

- CMake and g++
- Recommended IDE: **CLion**
  - https://download.cvut.cz, JetBrains
- Homework and semestral project skeletons provided only as Cmake projects
- See next slides for your platform

Windows+Visual Studio? :(

- Use at your own risk
- **Do not use MSVC** (no support for newer OpenMP)

Ubuntu toolchain

- You can use inofficial PPA for the Clion, see this link.
- Install g++ and cmake

```
>> sudo apt install g++ cmake [gdb]
```

- Install MPI library

```
>> sudo apt install libopenmpi-dev
```

Windows mingw toolchain

- Install msys2, see this link
- In the msys2 console do the following

```
>> pacman -Syu
>> pacman -Su
>> pacman -S base-devel mingw-w64-x86_64-toolchain
>> pacman -S mingw-w64-x86_64-msmpi
```

- Create MinGW toolchain in CLion, see this link. If msys2 is installed in default location, set `C:\msys64\mingw64` as your MinGW Environment path (everything else should be detected automatically), Setup in CLion Settings -> Cmake -> Generator on value MINGW Makefiles
- Add msys2 directories to your PATH environment variable, e.g.,

```
C:\msys64
C:\msys64\mingw64\bin
```

- If MPI library found, but program returns nonzero code and no output printed, try install this link, magically helped

Mingw CLion settings CMake

Settings

Build, Execution, Deployment › CMake

- Appearance & Behavior
- Keymap
- Editor
- Plugins
- Version Control
- Build, Execution, Deployment
  - Toolchains
  - CMake
  - Compilation Database
  - Custom Build Targets
  - Makefile
  - Build Tools
  - Debugger
  - Python Debugger
  - Python Interpreter
  - Deployment
  - Console
  - Coverage
  - Docker
  - Dynamic Analysis Tools
  - Embedded Development
  - Required Plugins

☐ Reload CMake project on editing CMakeLists.txt or other CMake configuration files

External changes, such as VCS update, always trigger project reload

Profiles

Profile is a named set of build options. For example, create separate profiles for Debug and Release builds and switch between them when needed.

Debug

☑ Enable profile

Name: Debug ☐ Share

Build type: Debug — Corresponds to CMAKE_BUILD_TYPE

Toolchain: Use default MinGW — Manage toolchains...

Generator: Use default MinGW Makefiles

CMake options: All CMake options

```
-G "CodeBlocks - MinGW Makefiles" -DCMAKE_BUILD_TYPE=Debug
```

Cache variables

Build directory: cmake-build-debug

Build options: -- -j 6

Arguments after '--' are passed to the build, other arguments are CMake command line parameters. Default options depend on the toolchain's environment.

OK Cancel Apply

Windows alternative WSL toolchain

- Install WSL, see this link
  - In powershell run: `wsl --install`
- Install Ubuntu distribution via microsoft store, see this link
- Open Ubuntu terminal, initiate system (user access setup, first run), install following
  - `sudo apt-get update`
  - `sudo apt install g++ cmake gdb`
  - `sudo apt install libopenmpi-dev`
- Set up WSL in Clion toolchains, see this link

WSL CLion settings Toolchains

Settings

Build, Execution, Deployment › Toolchains

- Appearance & Behavior
- Keymap
- Editor
- Plugins
- Version Control
- Build, Execution, Deployment
  - Toolchains
  - CMake
  - Compilation Database
  - Custom Build Targets
  - Makefile
  - Build Tools
  - Debugger
  - Python Debugger
  - Python Interpreter
  - Deployment
  - Console
  - Coverage
  - Docker
  - Dynamic Analysis Tools
  - Embedded Development
  - Required Plugins

WSL (default)
MinGW

Name: WSL — Add environment

Toolset: Ubuntu
Version: Ubuntu 20.04 LTS — Download...

CMake: WSL CMake
Version: 3.16.3

Build Tool: Detected: make
C Compiler: Detected: cc
C++ Compiler: Detected: c++

Debugger: WSL GDB
Version: 9.1

OK Cancel Apply

WSL CLion settings CMake

Settings

Build, Execution, Deployment › CMake

- Appearance & Behavior
- Keymap
- Editor
- Plugins
- Version Control
- Build, Execution, Deployment
  - Toolchains
  - CMake
  - Compilation Database
  - Custom Build Targets
  - Makefile
  - Build Tools
  - Debugger
  - Python Debugger
  - Python Interpreter
  - Deployment
  - Console
  - Coverage
  - Docker
  - Dynamic Analysis Tools
  - Embedded Development
  - Required Plugins

Reload CMake project on editing CMakeLists.txt or other CMake configuration files

External changes, such as VCS update, always trigger project reload

Profiles

Profile is a named set of build options. For example, create separate profiles for Debug and Release builds and switch between them when needed.

Debug

Enable profile

Name: Debug

Share

Build type: Debug — Corresponds to CMAKE_BUILD_TYPE

Toolchain: Use default WSL — Manage toolchains...

Generator: Let CMake decide — Don't pass any generator explicitly

CMake options: All CMake options

-DCMAKE_BUILD_TYPE=Debug

Cache variables

Build directory: cmake-build-debug

Build options:

Arguments after '--' are passed to the build, other arguments are CMake command line parameters. Default options depend on the toolchain's environment.

OK Cancel Apply

MacOS toolchains

- Using g++ **(recommended)**
  - Install g++ from Homebrew
    ```
    >> brew install gcc
    ```
  - Find the installed g++ executable. Usually a program called ***g++-FŇ*** where ***FŇ*** is the version (can be found using TAB completion), e.g., ***g++-9***
  - Set ***g++-FŇ*** compiler in CLion: Settings → Build, execution, Deployment →Toolchains → C++ compiler
- Using clang
  - Install OpenMP runtime from Homebrew
    ```
    >> brew install libomp
    ```
  - Check where libomp is installed, usually `/usr/local/opt/libomp`
    ```
    >> brew --prefix libomp
    ```
  - Link OpenMP into CMakeLists.txt
    ```
    include_directories("/usr/local/include" "/usr/local/opt/libomp/include")
    link_directories("/usr/local/lib" "/usr/local/opt/libomp/lib")
    ```
- Install MPI
  ```
  >> brew install open-mpi
  ```

Expected cmake console print for all

- Found OpenMP/MPI TRUE OPENMP (needed for next week), MPI (there is time to solve issues, used later)

CMake Debug

```
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- Found MPI_C: /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so (found version "3.1")
-- Found MPI_CXX: /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi_cxx.so (found version "3.1")
-- Found MPI: TRUE (found version "3.1")
-- Configuring done
-- Generating done
-- Build files have been written to: /mnt/c/Users/stejs/Desktop/week1_codes/cmake/cmake-build-debug

[Finished]
```

Expected program outputs

- Expected console outputs for provided helloworld programs to test your environment
- OpenMP

  Value of x: 550

  Process finished with exit code 0
- MPI

  My ranking hello world example: 0
  Total number of processes: 1

  Process finished with exit code 0