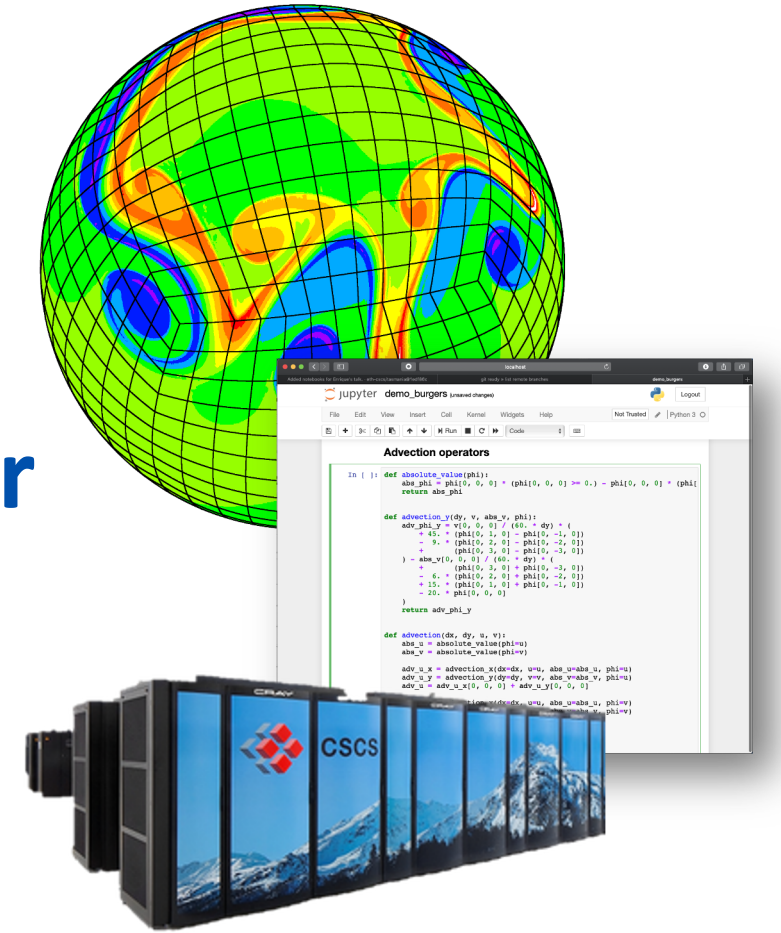
High Performance Computing for Weather and Climate (HPC4WC)

Content: Shared Memory Parallelism
Lecturers: Tobias Wicky
Block course 701-1270-00L
Summer 2020

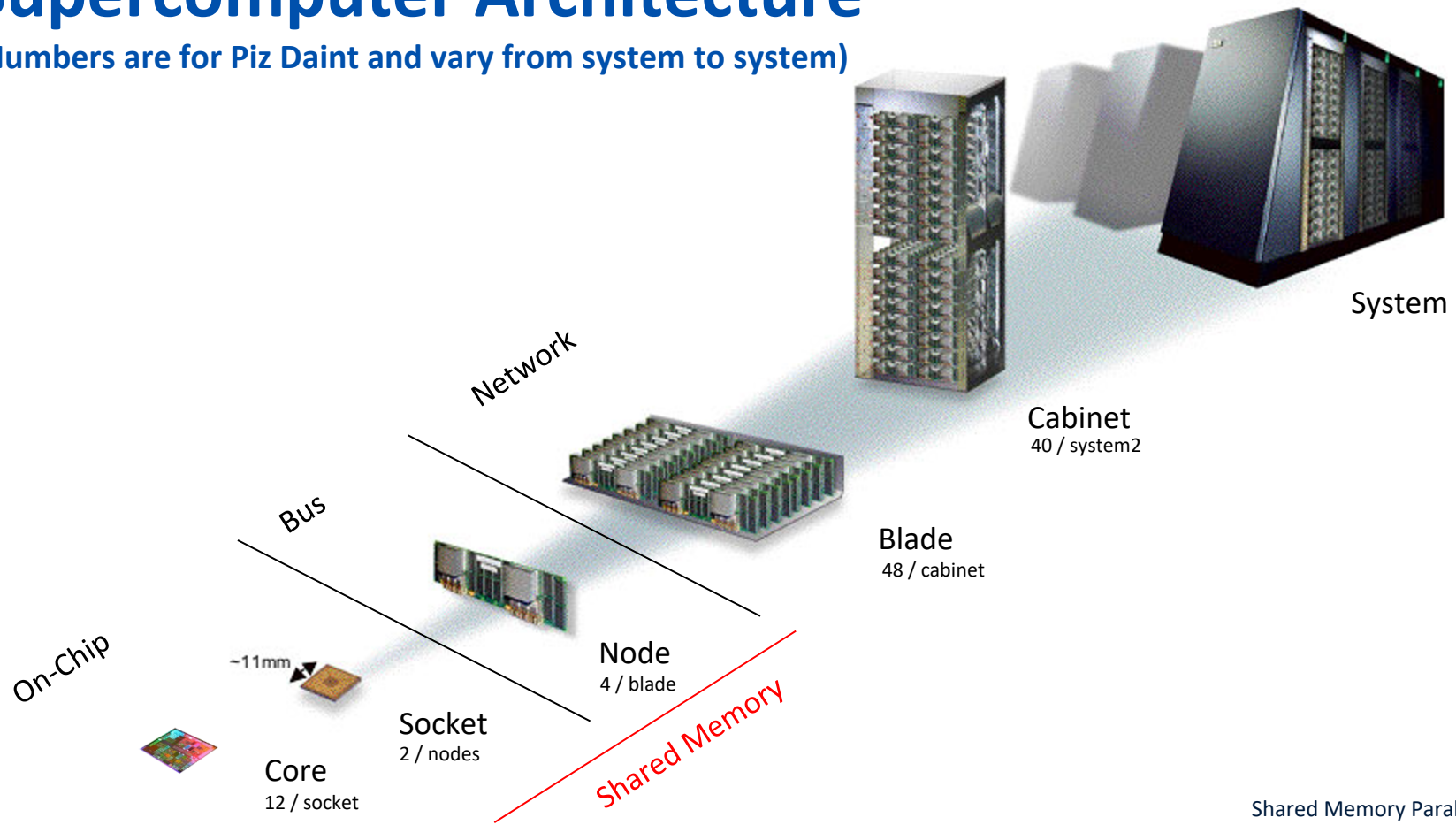


Learning goals

- Understand the shared memory parallelism and the OpenMP programming model
- Understand the limitation of parallelism with Amdahl's law
- Know about common pitfalls in shared memory computing

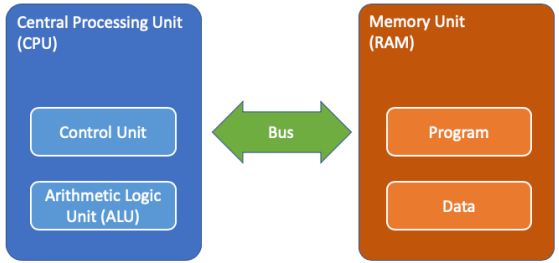
Supercomputer Architecture

(Numbers are for Piz Daint and vary from system to system)

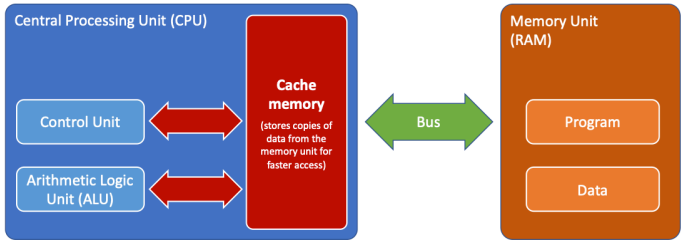


Node Architecture

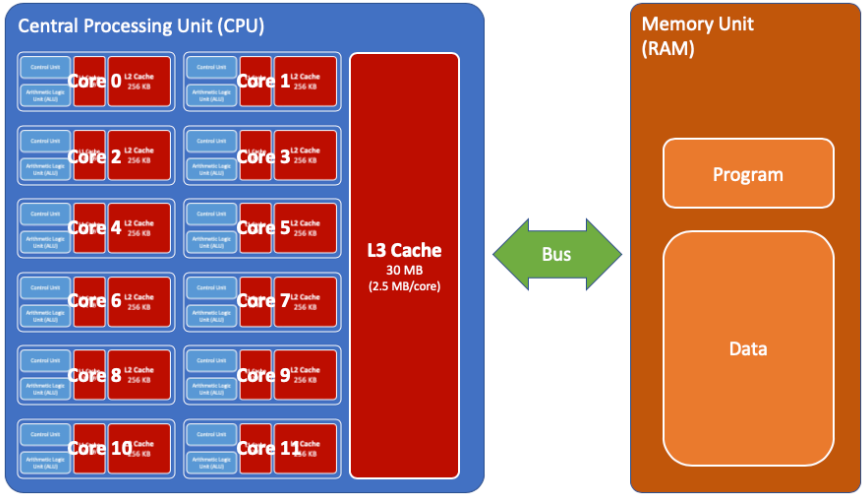
Von Neumann



Cache hierarchy



Multicore CPU

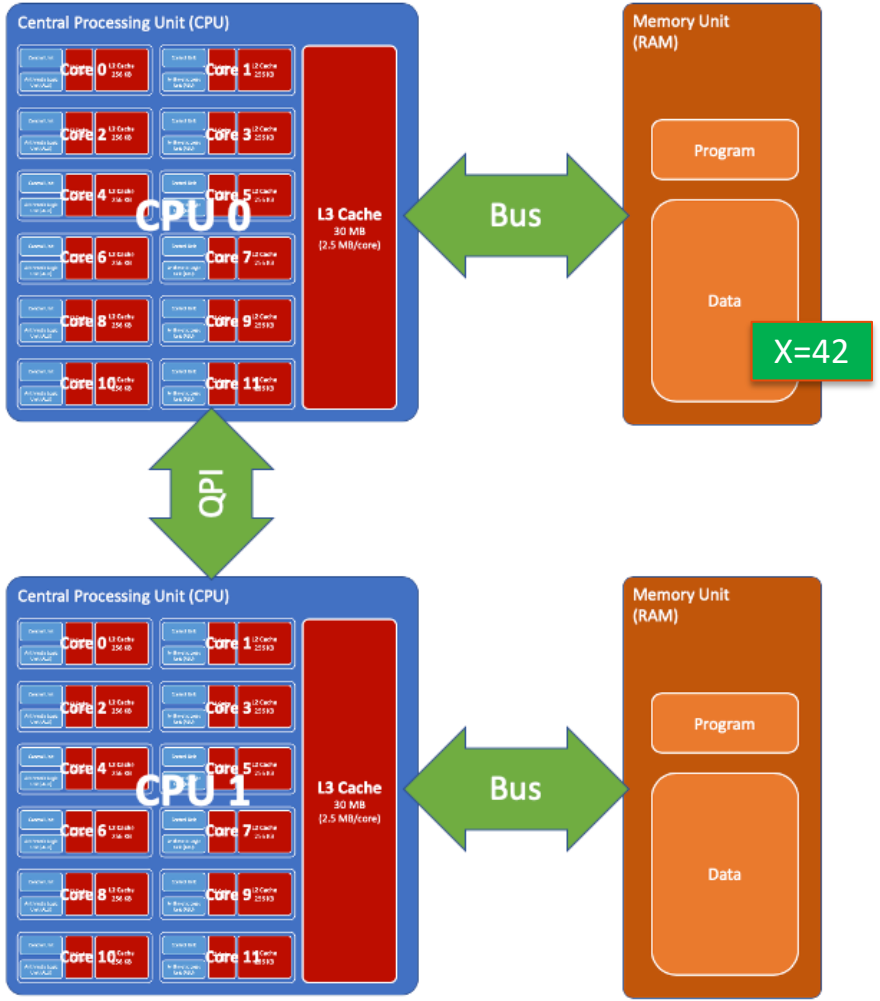


Node Architecture

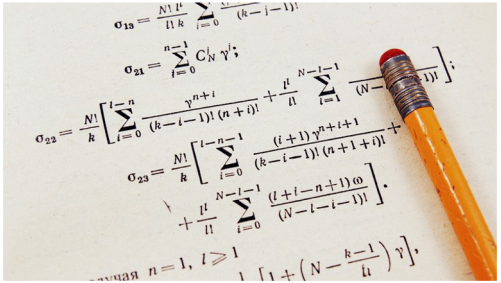
Share memory node

- Multiple CPUs (1, 2, 4, …)
- Connected via Bus (QPI)
- Many cores (12, 24, 36, …)
- Multiple memorys
- Shared address space (data in any memory is accessible to any core)

To make efficient use of the resources, a program must run in parallel on multiple cores.



Parallel Computing



Demo: Calculating Pi

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots = \frac{\pi}{4}$$

- Assume we have 12 cores, what can we do?
- How fast do we expect it to be?

OpenMP

- Open Multi-Processing is an API that supports shared-memory multiprocessing (https://www.openmp.org/)
- Version 1.0 in 1997, latest Version 5.0 in 2018
- Support for Fortran, C, C++
- Common programming model used in HPC
- Section of code that should run in parallel is marked with a compiler directive (if ignore, legal sequential code)
- Reference sheet of Fortran API v4.0

```c
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

```
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
```

Compiler directives

Pros

- semi automatic parallelisation
- portable across platforms & compilers

Cons

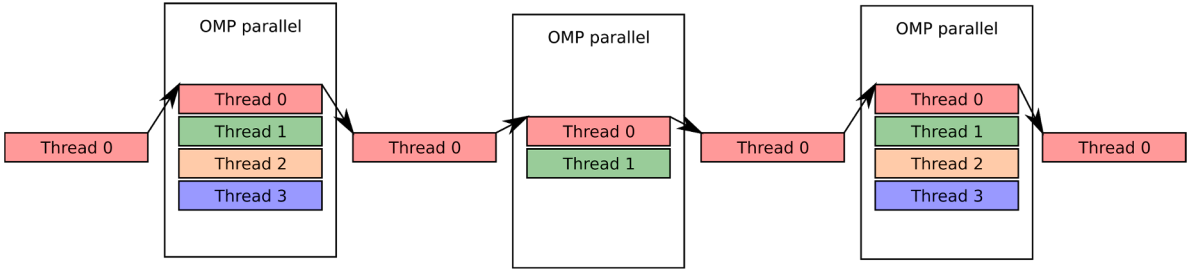
- non optimal code
- not the same for each compiler
- not safe

```
void spawnThreads(int n) {
  std::vector<thread> threads(n);
  for(int i = 0; i < n; i++) {
    threads[i] = thread(doSomething, i + 1);
  }

  for(auto& th : threads) {
    th.join();
  }
}
```

```
omp_set_num_threads(n)
#pragma omp parallel
  { do_something(omp_get_thread_num()); }
```

The fork-join model



- One master thread that runs through the full programm
- Parallel regions that can fork multiple processes that can be handled in parallel

Pitfalls in parallel computing: Race conditions

- If two processors can write at once the result is unknown

Pitfalls in parallel computing: Race conditions

“The” solution: Let only one processor write at once

How to do this?

- only one processor is in the region
- the variable is locked during my update
- write to my own variables

Solving the synchronisation issue

omp critical (name)	only one thread is ever in name
omp single	only one thread executes this code
omp master	only the master thread executes this code
omp ordered	the executions of the loop will do the ordered region sequentially
omp parallel schedule (type [,chunk])	enforces a schedule on the loop execution

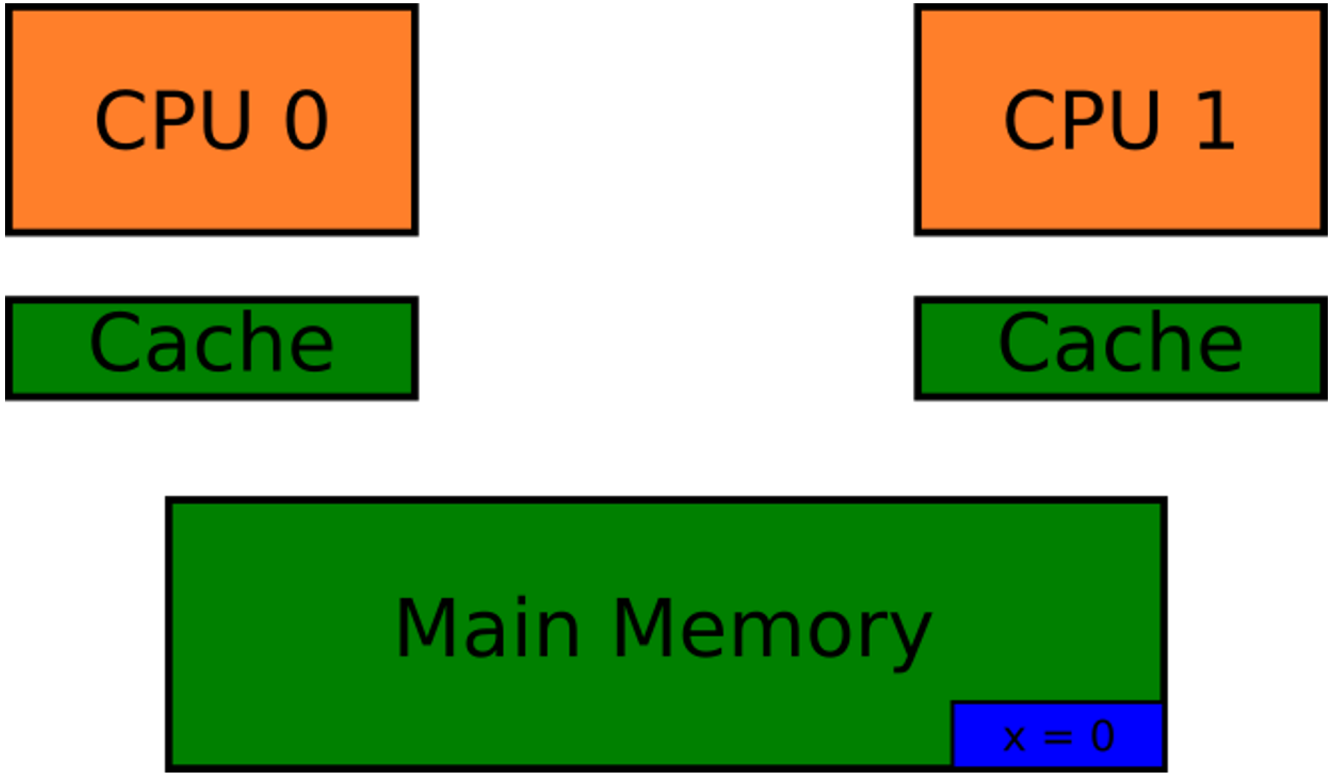
Solving the synchronisation issue

omp flush	writes the result to all threads
omp atomic	locks the variable while writing
private	every thread gets a copy of this variable
threadprivate	every thread gets a copy that persists

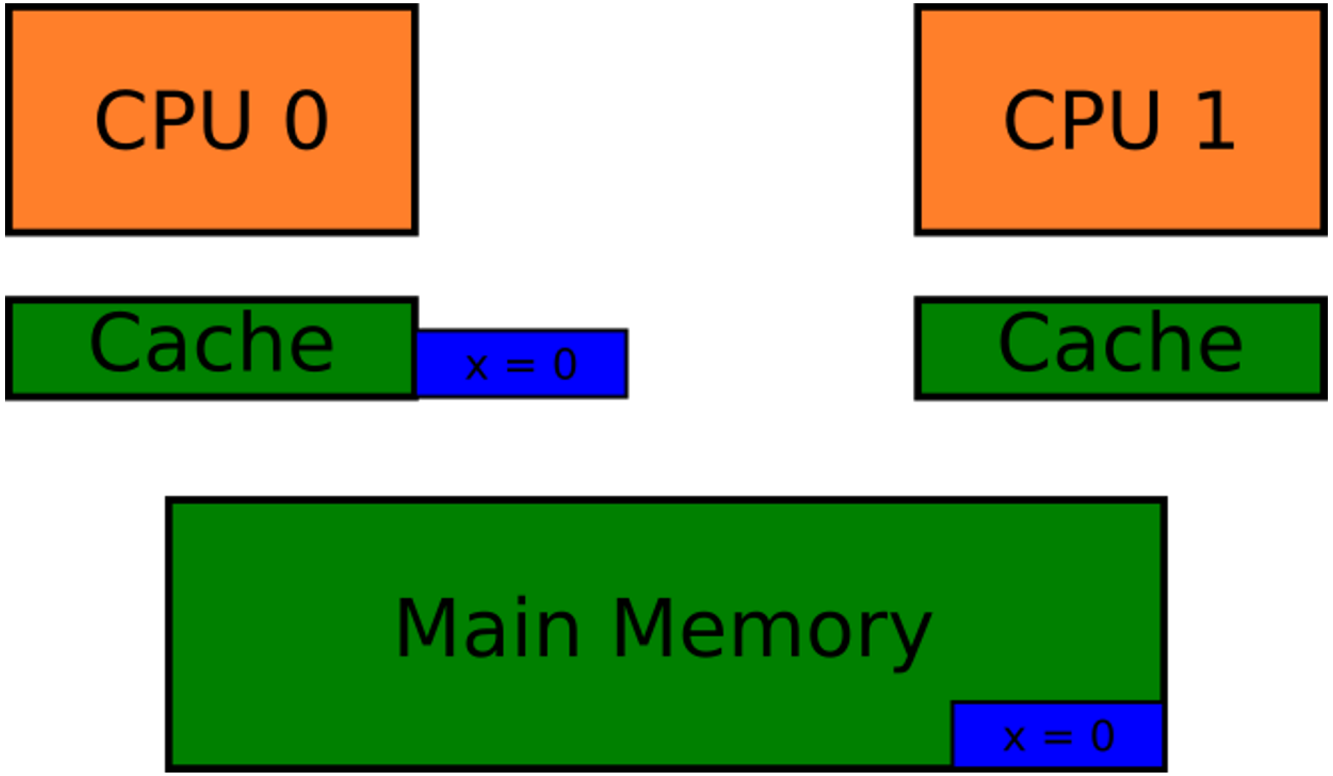
Pitfalls in parallel computing: Cache invalidation

- What happens if we store a variable in a cache?
  - what if it is used by more than one processor

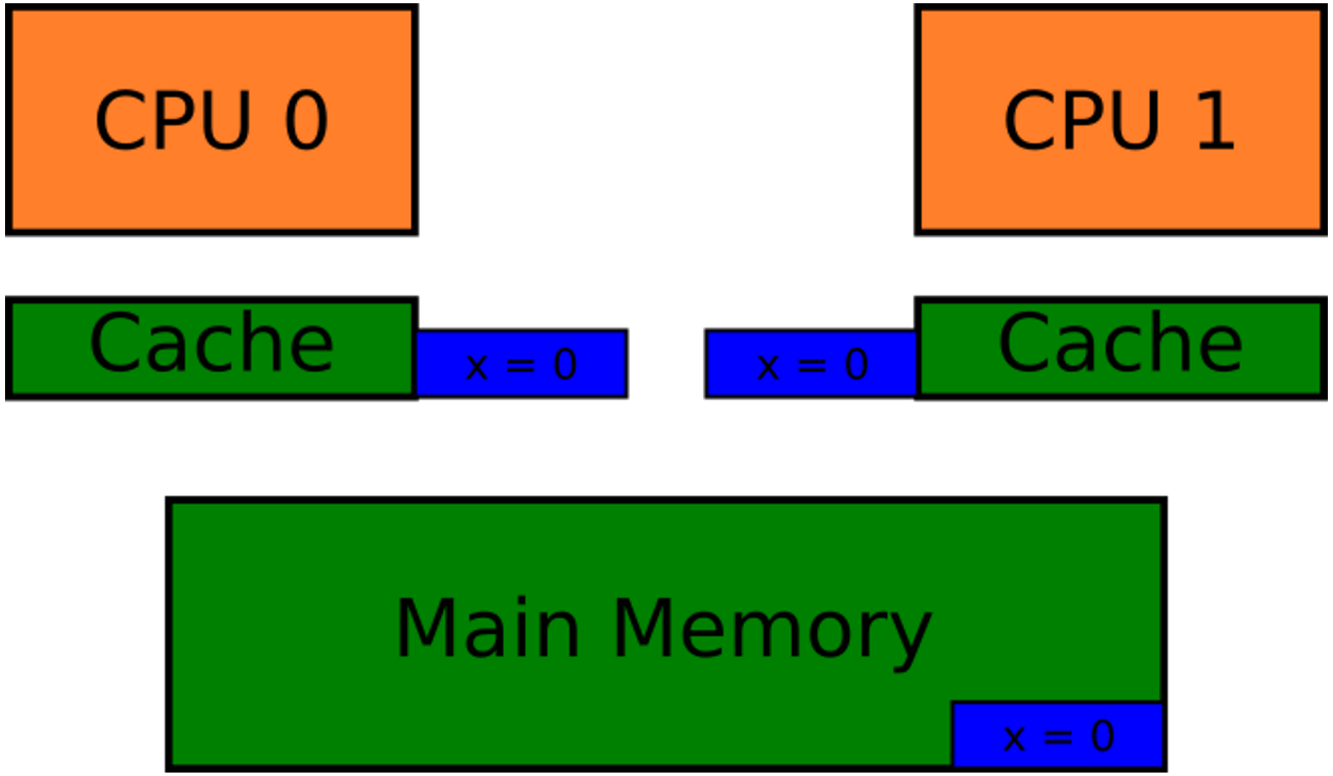
Cache invalidation



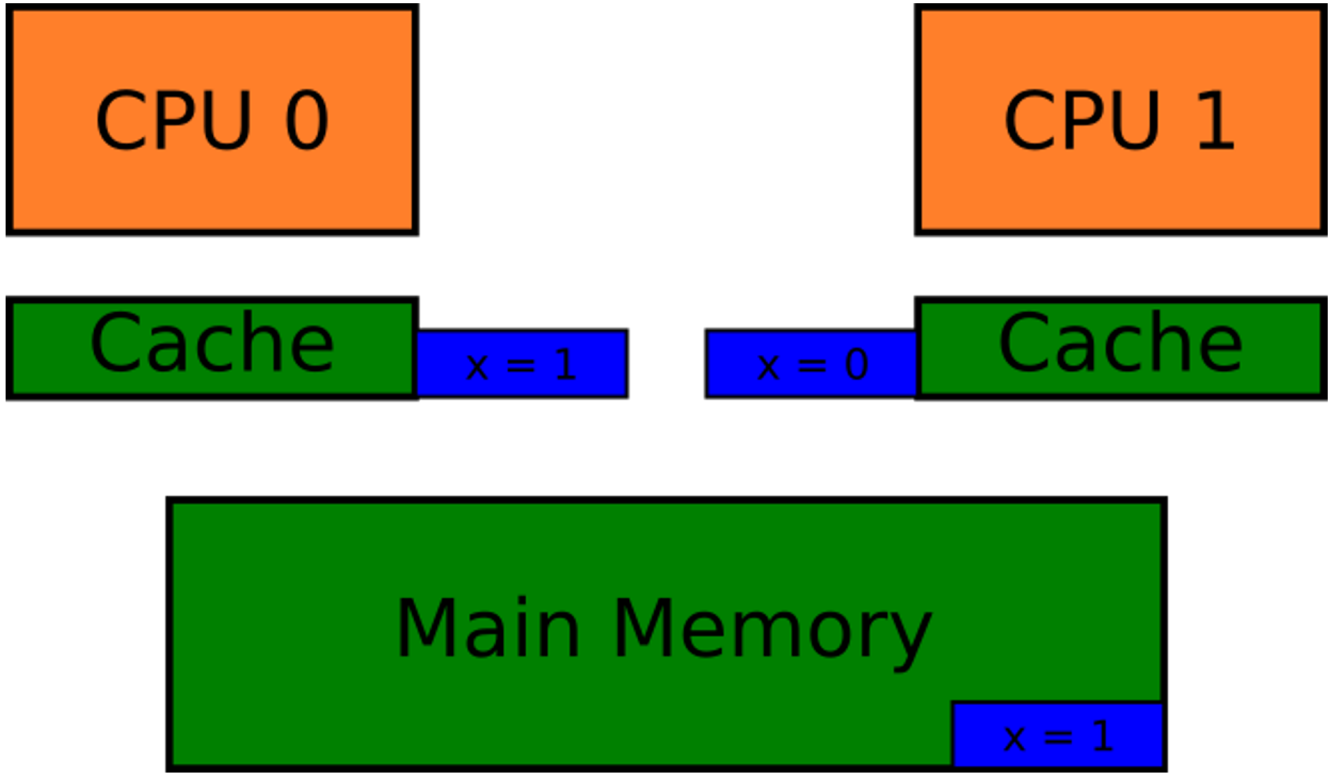
Cache invalidation



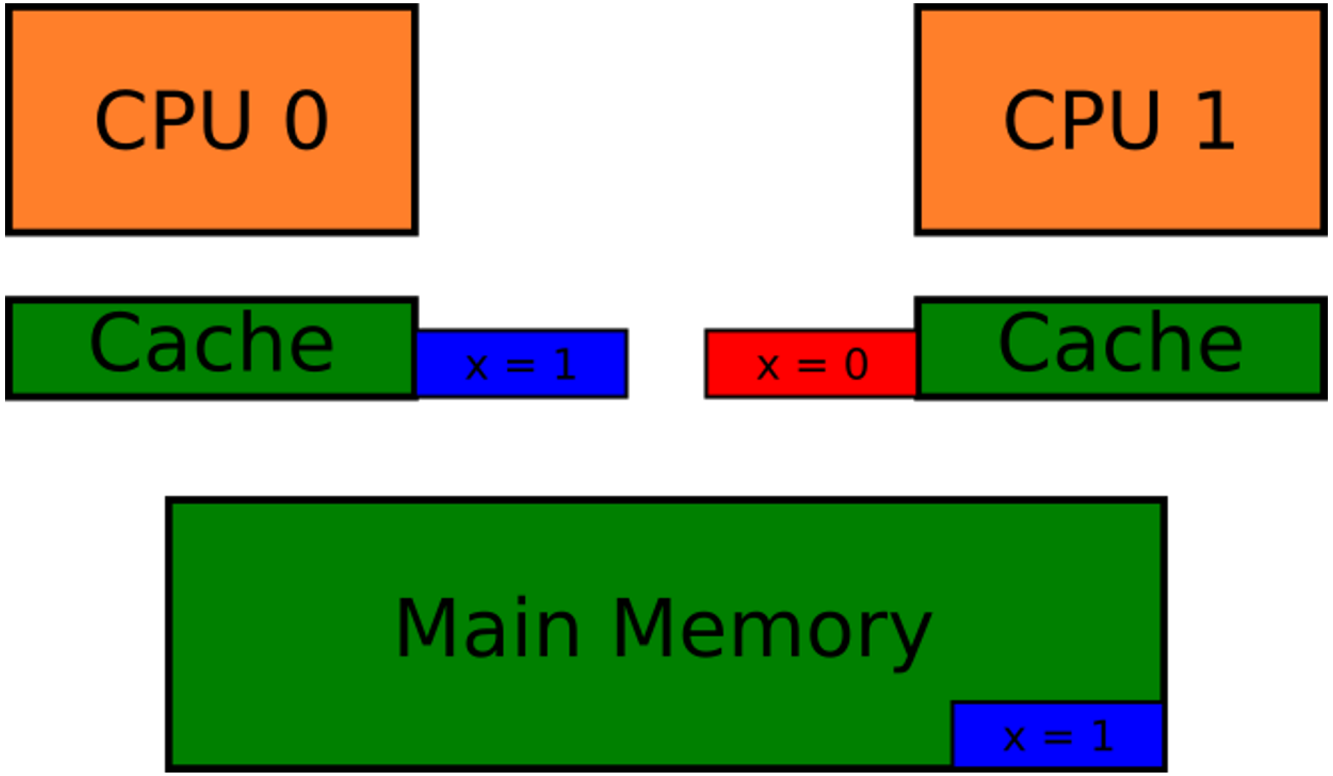
Cache invalidation



Cache invalidation



Cache invalidation



Performance implications

Most of the cache protocols right now take care of correctness for us

We need to ensure performance

Demo: Speedup for cache writes

Lab Exercises

01-OpenMP-introduction.ipynb

- Learn the basic OpenMP concepts (in C++, from lecture)

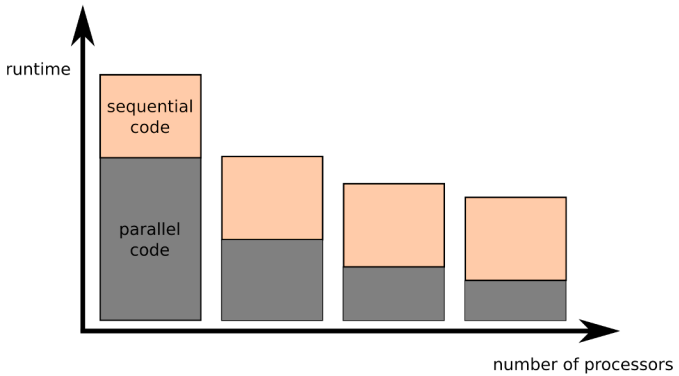
02-OpenMP-exercises.ipynb

- Parallelize the stencil2d program in Fortran using OpenMP
- Perform basic data-locality optimizations (fusion, inlining)
- Use a performance using a profiling tool for analysis and guidance

Note: Take a look at the OpenMP-Fortran-Cheatsheet.pdf to get help for how to use OpenMP in Fortran!

Amdahl's law

$$T(s) = (1-p)T_1 + \frac{p}{s}T_1$$

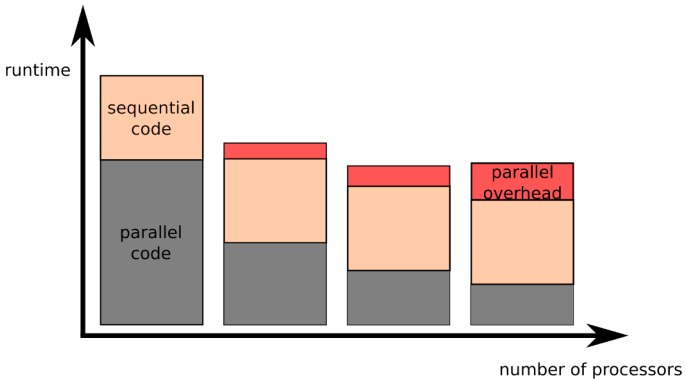


Sequential part of the code

- Opening the notebook
- Filling in the name

Amdahl's law

$$T(s) = (1-p)T_1 + \frac{p}{s}T_1$$

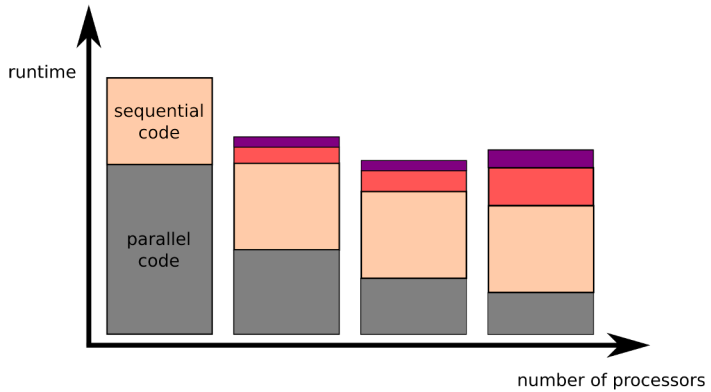


Parallel overhead

- making sure the order is correct
- synchronizing who does what

Amdahl's law

$$T(s) = (1-p)T_1 + \frac{p}{s}T_1$$



Load imbalance

- waiting for the other person to finish writing
- waiting for the other person to finish their task

Demo: performance check of various test cases