Handbook HPC Course

Martin Raum

September 1, 2020 at 11:10

This handbook, the website, and the course slides contain all material for the HPC course. The purpose of the handbook is to provide the organizational frame.

Part I.

Course Structure

1. Operation procedures

1.1. Lectures and demonstrations

1. Plan Zoom meeting. Acquire ID and password. No waiting rooms.
2. Post to Canvas.
3. Join 5 to 10 minutes early to avoid any technical problem.

1.2. Reading and watching

1. List of slide and videos links on website as part of student schedule.
2. Videos as files on Canvas?

1.3. Code counseling

1. Plan global Zoom meeting.
2. Post to Canvas.
3. Keep help folder in scratch. Update when students ask on Zoom.
4. Students can launch their own Zoom/Jitsi/Nextcloud sessions. When asking for help, need data to join.

1.4. Accounts

1. Send out login date with expiration 2 days.
2. Feedback from student to confirm participation; then extend expiration.

1.5. Assignments

1. First two assignments alone or in free groups.
2. Every coding lab has an initial 15 minutes breakout session, in which two students explain to each other what they have worked on and what they plan to work on in the lab. People with similar ambition can find each other this way.

3. Create random groups in the beginning of week three.
4. Let adjust if desired until end of week four. Keep record of groups from then on.
5. Extra groups can be created for bonus assignments.
6. Submit on Canvas (automatically via group tag).

2. Timeline and student schedule

2.1. Priorities by week

Course week 1 Gain familiarity with course structure. Train for distance work. Prepare for code counseling and programming. Start to learn about byte representation.

Course week 2 Consolidate understanding of byte representations. Learn to debug.

Course week 3 Optimization opportunities. Learn to benchmark.

Course week 4 Optimization opportunities. Learn to profile.

Course week 5 Learn thread parallelism.

Course week 6 OpenMP. If on optional track: Thread library.

Course week 7 Thread library. If on optional track: OpenCL.

Course week 8 Consolidate and prepare for exam. If on optional track: MPI.

2.2. Student tasks

2.2.1. Course week 1

Lecture Welcome.
Lecture Course organization.
Lecture Student tasks.
Lecture Log in to the training system gantenbein.
Lecture TMux Collaboration.

Website Course organization and student tasks.

Watch Command line usage and basic workflow.
Watch Basic programming.
Watch Memory in C.

Code counseling Command line usage and basic workflow.
Code counseling Basic programming.

Assignment Basic programming.

2.2.2. Course week 2

Lecture What is memory?
Lecture Debugging.
Lecture Best practice and common mistakes from the lab.

Read and watch What is HPC?
Watch What is HPC used for?
Watch Files in C and binary data
Watch Debugging.
Watch Hardware architecture.

Code counseling Basic programming.
Code counseling Debugging.

Assignment Basic programming.

2.2.3. Course week 3

Lecture Basic assembler.
Lecture Best practice and common mistakes from the lab.

Read and watch TOP500 HPC systems.
Watch Benchmarking.
Watch Basic assembler.
Watch Compiler based optimization
Watch Compilation units and code organization.
Watch Command line tools.

Code counseling Debugging.

Assignment Performance measurement, diagnosis, and basic optimization.

2.2.4. Course week 4

Lecture Profiling.
Lecture Best practice and common mistakes from the lab.

Watch Profiling.
Watch Performance impact of instructions
Watch Performance impact of cache locality.
Watch Basic parallel programming I.

Code counseling Debugging.

Assignment Performance measurement, diagnosis, and basic optimization.

2.2.5. Course week 5

Lecture Best practice and common mistakes from the lab.

Watch Vector instructions.
Watch OpenMP Classical parallelization.

Code counseling Debugging.
Code counseling Performance.

Assignment OpenMP.

2.2.6. Course week 6

Lecture Exam topics.
Lecture Best practice and common mistakes from the lab.

Read Website list of exam topics.
Watch Performance impact of instruction pipelines.
Watch Basic parallel programming II.
Watch OpenMP Tasks and dependencies.

Code counseling Debugging.
Code counseling Performance.

Assignment OpenMP.

If on optional track aiming to finish the assignments on both OpenCL and MPI then also adopt the following from course week 7.

Watch Thread library I.

Assignment Thread library.

2.2.7. Course week 7

Lecture Best practice and common mistakes from the lab.

Watch Thread library I.
Watch GPU programming I.

Code counseling Debugging.
Code counseling Performance.

Assignment Thread library.

If on optional track aiming to finish the assignments on both OpenCL and MPI then also adopt the following from course week 8.

Assignment OpenCL.

Watch GPU programming II.

2.2.8. Course week 8

Lecture Best practice and common mistakes from the lab.

Watch Thread library II.
Watch GPU programming II.
Watch Distributed computing.

Code counseling Debugging.
Code counseling Performance.

Assignment Thread library.

If on optional track aiming to finish the assignments on both OpenCL and MPI then also adopt the following from course week 8.

Watch HPC systems.

Assignment OpenCL.

Assignment MPI.

2.2.9. Exam

- Pick topic. Identify in sections on optimization, OpenMP, thread library a specific aspect to investigate. If on optional track: Can choose from OpenCL or MPI.
- Collect alternative implementations with regard to that aspect. Predict performance impact.
- Implement them in chosen assignment.
- Benchmark all of them. Compare with expectation.
- Write slides summarizing results.
- Practice presentation (best present to fellow students).
- Collect and study possible exam questions asked that connect to the presentation.

3. Topics

The topics in the course are yields a coarse order of skills and tasks, but do not reflect how the course is actually run. We do not assign numbers to the topic, since we want to avoid suggesting any order.

3.1. Introduction and Generalities

- Course organization and student tasks.
- Literature and material.
- Exam topics.
- Operation procedure.
- What is HPC?
- What is HPC used for?
- TOP500 HPC systems.

3.2. Basics: Command line, Editor, Programming

3.2.1. Command line usage and basic workflow

- Log in to the training system gantenbein.
- File system navigation.
- File system operations.
- Invoking commands.
- Text editor.
- Terminal multiplexer.
- TMux workflow.
- TMux Collaboration.

3.2.2. Command line tools

- grep.

- find.
- for.
- xargs.

3.2.3. Basic programming

- First C program.
- First makefile.
- Revisit control flow.
- Function calls.
- Reading cppreference.

3.2.4. Memory in C

- What is memory?
- Arrays.
- Memory allocation.
- Contiguous memory.
- Arrays as memory allocation.
- Pointer arithmetic.

3.2.5. Files in C and binary data

- File access.
- Text vs. binary files.
- String encoding and handling.

3.2.6. Compilation units and code organization

- Compilation Units.
- Include and define statements.
- make.

3.2.7. Debugging

- GDB.
- valgrind.
- GDB for parallel debugging.

3.2.8. Basic assembler

- Generate, read, debug assembler code.
- Assembler: If statements and for loops.
- Assembler: Integer vs. floating point.
- Assembler: Function calls.

3.2.9. Benchmarking

- Naive benchmarking.
- hyperfine.

3.2.10. Profiling

- gprof.
- Perf.
- cachegrind.

3.2.11. HPC systems

- Linker.
- PATH variables.
- Static and dynamic libraries.
- BLAS, LINPACK, GMP, FLINT.
- C3SE walkthrough incl. Slurm.

3.3. Optimization and hardware Architecture

3.3.1. Hardware architecture

- Simple machine model.
- CPU vs. memory bound code.

3.3.2. Compiler based optimization

- Compiler options.
- Inlining.

3.3.3. Performance impact of instructions

- Performed operation.
- Operand type.
- Operand value.
- Operand packing.

3.3.4. Performance impact of cache locality

- Locality.
- Indirect addressing.
- Padding and size of structures.
- File vs memory access.
- Cache replacement policy.

3.3.5. Performance impact of instruction pipelines

- Instruction pipeline
- Aliasing.
- Branching.

3.3.6. Vector instructions

- SIMD. RISC vs. CISC.
- Instrinsics.
- Assembler: Vector instructions.

3.4. Parallel Programming

3.4.1. Basic parallel computation

- Types of parallelism.
- Naive parallelization.
- Data partitioning.
- Time complexity and parallelization.
- Concurrency and deadlock.

3.4.2. OpenMP

- What is OpenMP?
- For loops and reduction.
- Attributes.
- SIMD loops.
- Parallel statements.
- Schedulers.
- Tasks.
- Dependencies.

3.4.3. Thread library

- Purpose of thread library.
- Thread management.
- Communication among threads.
- Example of parallelized multi-stage processing.
- Thread pool.

3.5. Accelerators and distributed computing

3.5.1. Beyond thread based computing

- Array parallelism.
- Stream parallelism.
- Distributed computing.

3.5.2. OpenCL

- What is OpenCL?
- Computation and memory model.
- Initialization and buffers.
- OpenCL programs and kernels.
- Example matrix multiplication.
- Vector types.
- Reduction on GPUs.

3.5.3. MPI

- What is MPI?
- Building and running a basic MPI program.
- Send and recv communication.
- Group based communication.
- Communicators.

Part II.

Work Units

4. Work units by presentation in course

4.1. Lectures and demonstrations (live)

4.1.1. Introduction and Generalities

1. Lecture Welcome. Welcome to Chalmers and course. Introduce teachers. Encourage mutual support and talk about diversity. Use Slides 1. Alternatively, watch short Video 1.

2. Course organization. Go through the main points of the organization. Use website list 1. Use Slides 1.

3. Students tasks. Go through the main points of student tasks. Use website list 2. Use Slides 1.

4. Exam topics. Go through the website list 4 of possible topics for the exam and explain what is expected to complete them.

4.1.2. Basics: Command line, Editor, Programming

1. Log in to the training system gantenbein. Revisit content of Video 1.

2. TMux Collaboration. Revisit content of Slides 6 and Video 8.

3. What is memory? Revisit content of Video 19.

4. Basic assembler. Revisit content of Video 32.

5. Debugging. Revisit content of Videos 29 and 30.

6. Profiling. Revisit content of Video 40.

4.1.3. Code counseling amendments

1. Best practice and common mistakes from the lab. Pick examples that showed up in the lab and discuss them.

4.2. Reading and watching

4.2.1. Introduction and Generalities

1. Course organization and student tasks. Read website Sections 1 and 2.
2. Literature and material. Read website Section 3.
3. List of exam topics. Read website Section 4
4. What is HPC? Watch Video 2.

   Optional: Skim the article "There's plenty of room at the Top: What will drive computer performance after Moore's law?" [if you can get access to it, and if not acknowledge the negative contriubtion of scientific "publishing" to science].

   Optional: Alternatively, skim the performance tables in the slides "Performance Engineering of Software Systems, Lecture 1" by Charles E. Leiserson.

   Optional: Skim the article "Scalability! But at what COST?" [which is freely available].
5. What is HPC used for? Watch Video 3.
6. TOP500 HPC systems. Watch video 4 and read slides 4.

4.2.2. Basics: Command line, Editor, Programming

1. Command line usage and basic workflow. Watch Videos 1, 2, 3, 4, 5, 6, 7, and 8.
2. Command line tools. Optional: Watch Videos 9, 10, 11, and 12.
3. Basic programming. Watch Videos 14, 15, 16, 17, and 18. Work through Code with gaps 1, 2, 3, and 4.
4. Memory in C. Watch Videos 19, 20, 21, and 22.
5. Files in C and binary data Watch Videos 23, 24, and 25.

6. Compilation units and code organization. Watch Videos 26 and 27. Optional: Watch Video 28.
7. Debugging Watch Videos 30 and 29.
8. Basic assembler. Watch Videos 32, 33, 34, and 35.
9. Benchmarking. Watch Video 36. Optional: Watch Videos 37 and 38.
10. Profiling. Watch Video 40. Optional: Watch Videos 39 and 41.

4.2.3. Optimization and hardware Architecture

1. Hardware architecture. Watch Videos 1 and 2. Familiarize with, optionally learn fully: Watch Videos 3 and 4.
2. Compiler based optimization. Watch Videos 5 and 6. Read Slides 5.
3. Performance impact of instructions. Watch Videos 7, 8, and 9. Read Slides 6.
4. Performance impact of cache locality. Watch Videos 10, 11, 12, 13, and 14. Read Slides 7 and 8.
5. Performance impact of instruction pipelines. Watch Videos 15, 16, and 17. Read Slides 9 and 10.
6. Vector instructions. Watch Videos 18, 19, and 20. Read Slides 11.

4.2.4. Parallel Programming

1. Basic parallel programming I. Watch Videos 1, 2, and 3.
2. Basic parallel programming II. Watch Videos 4 and 5, and 31 from Section 5.3.2.
3. OpenMP Classical parallelization. Watch Videos 6, 7, 8, 9, 10, and 11.
4. OpenMP Tasks and dependencies. Watch Videos 12 and 13.
5. Thread library I. Watch Videos 14, 15, and 16.
6. Thread library II. Watch Videos 17, 18, and 19.

4.2.5. Accelerators and distributed computing

1. GPU programming I. Watch Videos 1, 4, 6, 7, and 8.
2. GPU programming II. Watch Videos 2, 5, 9, and 10.
3. Distributed computing. Watch Videos 3, 11, 12, 13, 14, and 15.
4. HPC systems. Watch Videos 42, 43, 44, 45, and 46.

4.3. Code counseling

4.3.1. General help This also covers the first two assignments.
Command line usage and basic workflow.

- Prepare for fixing collaboration on tmux via screen share.

Basic programming.

- Request that students watch Videos 14, 15, and 16.
- Give analogy to other languages incl. Python and Julia.
- Point to `https://projecteuler.net`.

Debugging.

- Request that students watch Videos 29 and 30.
- Assist with their concrete problems in all assignments.

Performance.

- Check algorithm.
- Check data types.
- Check locality.
- Check pipeline.
- Run perf.
- Assist with their concrete problems in all assignments.

4.3.2. Assignments Hints for the Assignments 4.4.3, 4.4.4, 4.4.5, and 4.4.6 are in the solution folders.

4.4. Assignments

4.4.1. Basic programming This assignment is an assortment of subtasks.

1. Stack vs. heap allocation.
2. Memory fragmentation.
3. Writing to files.
4. Parse command line arguments.
5. Naive benchmarking.

Most relevant work topics are:

- Basic programming.
- Memory in C.
- Files in C and binary data.

4.4.2. Performance measurement, diagnosis, and basic optimization This assignment is an assortment of subtasks.

1. Assembler.
2. Valgrind.
3. GDB.
4. Inlining.
5. Locality.
6. Data dependency.
7. Indirect addressing
8. HDD and SSD.

Most relevant work topics are:

- Basic assembler.
- Compiler based optimization.
- Compilation units and code organization.
- Debugging.
- Benchmarking.

4.4.3. OpenMP This assignment is one small project targeting basic parallelization via OpenMP, locality, string parsing, and at option SIMD operations.

Most relevant work topics are:

- Compiler based optimization.
- Performance impact of cache locality.
- Vector instructions.
- Basic parallel computation.
- OpenMP.

4.4.4. Thread library

Provide Julia reference implementation of Newton iteration.

This assignment is one small project targeting concurrency and synchronization in parallel programs, the thread library, instruction pipelines, and efficient string handling.

Most relevant work topics are:

- Compiler based optimization.
- Performance impact of instruction pipelines.
- Thread library.

4.4.5. OpenCL This assignment is one small project targeting the OpenCL basics, command queues, and host-device memory transfer.

Most relevant work topics are:

- Performance impact of cache locality.
- Beyond thread based computing.
- OpenCL.

4.4.6. MPI This assignment is one small project targeting the MPI basics and orchestration of progresses.

Most relevant work topics are:

- Performance impact of cache locality.
- MPI.

5. Work units by material

5.1. Website

1. Overview of course organization. Schedule, registration, lectures, demonstration, computer lab, exam.
2. Student task list.
   keep updated
3. Main literature and source of information. Sterling-Anderson-Brodowicz, High Performance Computing. Stackoverflow, Youtube videos.
4. List of exam topics.

5.2. Slides and Code

5.2.1. Introduction and Generalities

1. Welcome. Combined slides for welcome lecture, course organization, and student tasks.
2. What is HPC? Common definitions, Wirth's law, Chalmers program description, supercomputer vs. cloud, examples.
3. What is HPC used for? EuroHPC statement, bright and dark side, climate simulations, cancer detection, LHC data analysis.
4. TOP500. Top 5 systems (architecture, interconnect), Linux based, accelerators, interconnects, C3SE.

5.2.2. Basics: Command line, Editor, Programming

Command line usage and basic workflow

1. Log in to the training system gantenbein. Slides for Video 1.
2. File system. Slides for Videos 2 and 3.
3. Invoking commands. Slides for Video 4.
4. Text editor. Slides for Video 5.
5. Terminal multiplexer. Slides for Videos 6 and 7.
6. TMux Collaboration. Slides for Video 6.

Command line tools

write slides and code

7. tar. Code for Video 13.

Basic programming

1. First C program. Code for Video 14 incl. a variant with gaps.
2. First makefile. Code for Video 15 incl. a variant with gaps.
3. Revisit control flow. Code for Video 16 incl. a variant with gaps.
4. Function calls. Code for Video 17 incl. a variant with gaps.

Memory in C

5. What is memory? Slides Video 19.
6. Illustrate hexadecimal system in Julia. Code for Video 19.
7. Illustrate size of types and variables. Code for Video 19.
8. Illustrate array and heap allocation. Code for Video 19.
9. Arrays. Code for Video 20
10. Memory allocation. Code for Video 21.
11. Pointer arithmetic. Code for Video 22.

Files in C and binary data

12. File access. Code for Video 23.
13. Text vs. binary files. C Code for Video 24.
14. Text vs. binary files. Bash code xxd usage. Code for Video 24.
15. String encoding and handling. Code for Video 25.

Compilation units and code organization

16. Compilation Units. Slides for Video 26.
17. Compilation Units. Code for Video 26.
18. Include and define statements. Code for Video 27.
19. make. Code for Video 28.

Debugging

20. GDB. Slides for Video 29.
21. GDB. Code for Video 29.
22. valgrind. Code for Video 30.

Benchmarking

23. Naive benchmarking. Code for Video 36.
24. Celero. Code for Video 37.
25. hyperfine. Code for Video 38.

Basic assembler

26. Assembler. Code for Videos 32, 33, 34, and 35.

HPC systems

write slides and code

27. Linker. Code for Video 42.
28. PATH variables. Code for Video 43.
29. Static and dynamic libraries. Code for Video 44.
30. HPC standard libraries. Slides for 45.
31. C3SE walkthrough. Slides for 46.

5.2.3. Optimization and hardware Architecture

Hardware architecture

1. Simple machine model. Slides for Video 1.
2. CPU vs. memory bound code. One example each of slow code with too many instructions, branching, bad locality because of access, bad locality because of structure size. Slides for Video 2.
3. Memory hierarchy. Slides for Video 3.
4. CPU pipeline. Slides for Video 4.

Compiler based optimization

5. Compiler based optimization. Code for Video **??**.
6. Inlining. Code for Video 6.

Performance impact of instructions

write slides and code

7. Performance impact of instructions. Measurements. Display measurements for operations, operand type, operand value.

Performance impact of cache locality

write slides and code

8. Performance impact of cache locality. Display measurements for access order, indirect addressing, padding, file access.
9. Cache replacement policy. Slides for Video 14.

Performance impact of instruction pipelines

write slides and code

10. Instruction pipeline. Slides for Video 15.
11. Performance impact of instruction pipelines. Display measurements for aliasing, restrict, branching, loop unrolling.
12. Performance impact of vector instructions. Display measurements for 1, 2, 4, and 8 packed addition.

5.2.4. Parallel Programming

Basic parallel computation

write slides and code

1. Types of parallelism. Explain what threads share. Asynchronous means that you need synchronization.
2. Types of parallelism. Illustrate thread and process parallelism in Julia. HTop to visualize. Shared memory. Synchronization.
3. Naive parallelization. Use Julia atomics, reentrant lock, shared array, and channel.
4. Data partitioning. Use Julia threads and thread pools.
5. Time complexity and parallelization.
6. Concurrency and deadlock. Use Julia channels to illustrate deadlock.

OpenMP

write slides and code

7. What is OpenMP?
8. What is OpenMP?
9. For loops and reduction. Performance measurements.
10. For loops and reduction.
11. Attributes.
12. SIMD loops. Performance measurements.
13. SIMD loops.
14. Parallel statements.
15. Schedulers. Performance measurements.
16. Schedulers.
17. Tasks. Performance measurements.
18. Tasks.
19. Dependencies.

Thread library

write slides and code

20. Purpose of thread library.
21. Thread management.
22. Communication among threads.
23. False sharing.
24. Thread pool.
25. Example of parallelized multi-stage processing.

5.2.5. Accelerators and distributed computing

Beyond thread based computing

write slides and code

1. Array parallelism. Slides for Video 1.
2. Stream parallelism. Slides for Video 2.
3. Distributed computing. Slides for Video 3.

OpenCL

write slides and code

4. What is OpenCL? Slides for Video 4.
5. What is OpenCL? Code for Video 4.
6. Computation and memory model. Slides for 5.
7. Initialization and buffers. Code for Video 6.
8. OpenCL programs and kernels. Code for Video 7.
9. Example matrix multiplication. Code for Video 8.
10. Vector types. Code for Video 9.
11. Code for Video 10.

MPI

write slides and code

12. What is MPI? Slides for Video 11.
13. What is MPI? Code for Video 11.
14. Building and running a basic MPI program. Code for Video 11.
15. Send and recv communication. Slides for 13.

16. Send and recv communication. Code for Video 13.
17. Group based communication. Slides for Video 14.
18. Group based communication. Code for Video 14.
19. Communicators. Slides for Video 15.
20. Communicators. Code for Video 15.

5.3. Video

5.3.1. Introduction and Generalities

1. Welcome. Explain briefly the role of videos and lectures.
2. What is HPC. Go through Slides 2.
3. What is HPC used for. Go through Slides 3.
4. Present some features of TOP500. Go through Slides 4

5.3.2. Basics: Command line, Editor, Programming

Command line usage and basic workflow

1. Log in to the training system gantenbein. First login, change of password, and exit of ssh session. Use Slides 1.
2. File system navigation. pwd, cd, ., .., ls. Use Slides 2.
3. File system operations. mv, cp, rm, mkdir. Use Slides 2.
4. Invoking commands. Command line history. Relative to path variables. Absolute path for executables. Getting help. Run arbitrary scripts. Execution privilege. Use Slides 3.
5. Text editor. Emacs: start, navigation, save, close, automatic backups. Use Slides 4.
6. Terminal multiplexer. tmux: start, detach, logout/login attach. Use Slides 5.

7. TMux workflow. split pane, select pane, rotate pane; text editor and shell to find compiler errors; text editor and two shells to find program errors.

8. TMux Collaboration. Create TMux with session file. Change access rights on session file. Connect to session file. Use Slides 6.

Command line tools

make videos

9. grep. Find text in one file. Find files with text. Print context.

10. for. Iteration through numbers, folder.

11. find. Name, filetype restriction. Execute combined with grep or sed (not explained in detail).

12. xargs. Parallel processing. Split up arguments.

13. tar. Create, extract, inspect tar files. Compression. Purpose. gzip and xz. Use Code 7.

Basic programming

14. First C program. main function incl. meaning of its signature; printf and stdio; gcc with output flag. Use Code 1.

15. First makefile. Make file that builds first C program; all, clean; PHONY. Use Code 2.

16. Revisit control flow. if, else, for, break, continue. Use Code 3.

17. Function calls. declaration, definition, return, arguments. Use Code 4.

18. Reading cppreference. Illustrate how to find information on printf.

Memory in C

19. What is memory? Variables need memory. Information is bits/bytes interpreted through type. First memory model: Sequence of bytes. Stack vs. heap. Use Slides 5 and Code 6, 7, and 8.

20. Arrays. Declaration and indexing of one-dimensional arrays. Multi-dimensional arrays. Arrays represent Contiguous memory. Pointers to members in the array. Use Code 9.

21. Memory allocation. Declaration, allocation, freeing, and indexing of point variables. Contiguous memory and memory fragmentation. Pointers to variables. Use Code 10.

22. Pointer arithmetic. Add to and subtract from pointer. Dereference. Conversion to char. Use Code 11.

Files in C and binary data

23. File access. Open and close files. Mode read, write, append. File descriptors can run out. Write to file and read from file. Jump inside of file. Functions fopen, fclose; fwrite, fread, fflush; fseek, ftell. Use Code 12.

24. Text vs. binary files. Illustrate binary file and text file with xxd. Write/print to file and read/parse from file. Functions fprintf, fscanf. Use Code 13 and **??**.

25. String encoding and handling. Null-terminated byte strings, conversion, strcpy and strcat, memset, memcpy, memmove. Use Code 14.

Compilation units and code organization

make videos

26. Compilation Units. Define notion. Illustrate in code. Use Slides 15 and Code 16.

27. Include and define statements. Preprocessor as opposed to compiler. Copy a file/-text verbatim, no syntax checked. Use Code 17.

28. make. Split up by compilation units. Make sure to track headers. Automatic variables. Use Code 18.

Debugging

make videos

29. GDB. Assertions. TUI: src. Print values. Breakpoints. Step through code. Traverse call stack. Command line arguments. Use Slides 19 and Code 20.
30. valgrind. Access violations. Find memory leaks. Volatile keyword to avoid optimization. Use Code 21.
31. GDB for parallel debugging. Thread view. Interaction of variables.

    make videos

Benchmarking

make videos

32. Naive benchmarking. Time and repeat function. Discuss impact of dead store elimination. Use Code 22.
33. Celero. Illustrate. Discuss difficulty of micro-benchmarks. Mention installation. Use Code 23.
34. hyperfine. Illustrate. Minimal runtime for meaningful results. Use Code 24.

Basic assembler

make videos

35. Generate, read, debug assembler code. Use Code **??**.
36. Assembler: If statements and for loops. Use Code **??**.
37. Assembler: Integer vs. floating point. Use Code **??**.
38. Assembler: Function calls. Use Code **??**.

Profiling

make videos

39. gprof. Coarse overview of hotspots. Interpret result.
40. Perf. Fine overview. Record different events: cache misses, branch misses. Interpret results.
41. cachegrind. Fine overview/simulation. Illustrate on example, interpret result. Mention callgrind.

HPC systems

make videos

42. Linker. Explain linker symbols, external linkage, link time optimization. Showcase ldd and nm. Use Code 25.
43. PATH variables. Illustrate with executables, include, and library paths. Use Code 26.
44. Static and dynamic libraries. Build shared and dynamic library. Illustrate performance difference. Use Code 27.
45. HPC standard libraries. Examples BLAS, LINPACK, GMP, NTL, FLINT. Use Slides 28.
46. C3SE walkthrough. Illustrate loading modules, building software, Slurm MPI, Slurm Julia. Use Slides 29.

5.3.3. Optimization and hardware Architecture

Hardware architecture

make videos

1. Simple machine model. CPU, memory, IO, network. Buses. Memory hierarchy. Use Slides 1.
2. CPU vs. memory bound code. To many instructions, too slow instructions (for instance memory access). Illustrate in one example each Use Code 2.

3. Memory hierarchy. Memory bandwidth and latency. Cache and memory hierarchy; CPU affinity. Examples. Visualization. Use Slides 3.

4. CPU pipeline. Instruction execution. Filled and broken pipelines. Examples. Use Slides 4.

Compiler based optimization

make videos

5. Compiler options. Optimization levels, architectures. GCC optimization options. Link time optimization. Profiler guided optimization. Use Code **??**.

6. Inlining. Call overhead. Inlining opportunities and statements.

Performance impact of instructions

make videos

7. Performed operations. Performed operation. Compare add, mul, div, sqrt. Use Slides 6.

8. Operand type. Compare sqrtf, sqrt. Mention rsqrt. Use Slides 6.

9. Operand value. Compare sin, exp with different argument size. Use Slides 6.

Performance impact of cache locality

make videos

10. Locality. Define memory locality. Illustrate access order in one, two, and three dimensions. Use Slides 7.

11. Indirect addressing. Explain direct and indirect addressing. Illustrate difference with shuffled vs. linear indexing. Use Slides 7.

12. Padding and size of structures. Illustrate padding in structures, and impact on locality. Use Slides 7.

13. File vs. memory access. Compare file and memory latency and bandwidth. Repeated reading zero bytes. Use Slides 7.

14. Cache replacement policy. Explain associative cache and TLB. Use Slides 8.

Performance impact of instruction pipelines

make videos

15. Instruction pipeline. Explain steps in pipeline and stale pipeline. Illustrate via data dependency in for loop. Use Slides 9 and 10.
16. Aliasing. Illustrate aliasing. Restrict keyword. Use Slides 10.
17. Branching. Illustrate branching. Unrolling short for loops. Branch prediction. Use Slides 10.

Vector instructions

make videos

18. SIMD. RISC vs. CISC. Explain instruction parallelism and vectorization. Throttling on Intel. Characteristics and examples of RISC and CISC architectures. Showcase lscpu and /proc/cpuinfo.
19. Instrinsics. Illustrate load, store, addition. Compare add on 1, 2, 4, and 8 ints. Mention auto-vectorization. Use Slides 11.
20. Assembler: Vector instructions. Illustrate assembler code for vector instructions.

5.3.4. Parallel Programming

Basic parallel programming

make videos

1. Types of parallelism. SISD, SIMD, MIMD. Processes and threads. Shared memory. Distributed computing. Use Code 2.
2. Naive parallelization. Dot product split across threads. Addition via atomics, via locks, and summation in main thread. Use Code 3.
3. Data partitioning. Sum up rows of triangular matrix. Naive block static, optimal block static, interleaved static, dynamic partitioning. Use Code 4.
4. Time complexity and parallelization. Sequential, parellizable, and communication time. Theoretically optimal speedup and superlinear speedup. Use Slides 5.
5. Concurrency and deadlock. Competition for resources incl. locks. Use Code 6.

OpenMP

make videos

6. What is OpenMP? Purpose and scope. Showcase for, tasks, and offloading incl. speedups. Use Slides 7 and Code 8.
7. For loops and reduction. Parallel for constructions. Reduction, critical, and atomic. Collapse (mention variable overflow). Locality. Use Slides 9 and Code 10.
8. Attributes. Shared and private. Firstprivate, lastprivate. Initialization of values. Functions. Use Code 11.
9. SIMD loops. Array reduction and SIMD. Use Slides 12 and Code 13.
10. Parallel statements. Parallel section, single, master, barriers, nowait, copyprivate. Use Code 14.
11. Schedulers. Static, dynamic, guided. Use Slides 15 and Code 16.
12. Tasks. Taskwait, taskgroup, Taskloop. Use Slides 17 Code 18.
13. Dependencies. Ordered, depend. Use Code 19.

Thread library

make videos

14. Purpose of thread library. C11 thread library. Purpose, limits, and liabilities. Use Code 20.
15. Thread management. Create, join, mutex, conditional variables. Use Code 21.
16. Communication among threads. Communication via shared memory constructions. Arrays and queues. Use Code 22.
17. Impact of false sharing. Use Code 23.
18. Example of parallelized multi-stage processing. Use Code 25.
19. Thread pool. FLINT thread pool usage and implementation. Use Code 24.

5.3.5. Accelerators and distributed computing

Beyond thread based computing

make videos

1. Array parallelism. Example of GPU and TPU. Branch divergence as main issue. Cache hierachy different from CPU. Use Slides 1.
2. Stream parallelism. Example of FPGA. Pipelining as design goal. Role of vector instruction. Registers as predominant locality notion. Use Slides 2.
3. Distributed computing. Example of super computing network and cloud. Non-uniform memory access and communication as main issue. Preemptive instances. Use Slides 3.

OpenCL

make videos

4. What is OpenCL? Explain history, compare to CUDA. Capabilities and implementations in various languages. Compare performance with dot product example. Use Slides 4 and Code 5.
5. Computation and memory model. Computation and memory hierarchy with all levels. Analogy to CPU and implementation in hardware. Match organization on CPU, GPU, and FPGA. Use Slides 6.
6. Initialization and buffers. OpenCL data types. Use Code 7.
7. OpenCL programs and kernels. Use Code 8.
8. Example matrix multiplication. Use Code 9.
9. Vector types. Illustrate speed difference on dot product. Use Code 10.
10. Reduction on GPUs. Use Code 11.

MPI

make videos

11. What is MPI? Explain history. Alternatives in big data. Capabilities and implementations in various languages. Performance with dot product. Use Slides 12 and Code 13.

12. Building and running a basic MPI program. Example of dot product. Use Code 13.

13. Send and recv communication. Example of ping pong communication. Model of central communication hub. Non-blocking variants. Impact of network topology. Use Slides 15 and Code 16.

14. Group based communication. Example of configuration broadcast. Example of vector accumulation. Use Slides 17 and Code 18.

15. Communicators. Computational domains, network cliques. Use Slides 19 and Code 20.

Chalmers tekniska högskola och Göteborgs Universitet, Institutionen för Matematiska vetenskaper, SE-412 96 Göteborg, Sweden
E-mail: martin@raum-brothers.eu
Homepage: http://raum-brothers.eu/martin