1. Introduction

1. 1. The challenge of big data

1. 1. 1. scenarios:

1. 1. 2. Intelligent CAD and CAE

1. 2. Modern PC hardware

1. 2. 1. Increasing computation power by Moore's law

1. 2. 2. Quantum computer

1. 3. Interaction with physical world

1. 3. 1. design prototype, CAD

1. 3. 2. optimisation of existing models

1. 3. 3. digital twin

1. 4. Are you using software correctly?

1. 4. 1. integer arithmatic overflow

1. 4. 2. process the data properly

1. 5. Audience of this book

2. Infrastructure for high-performance computation

2. 1. Introduction of compuation system

2. 1. 1. Choice of HPC resoruce

2. 1. 2. classic computation system arch

2. 1. 3. discusse the hardware bottleneck

2. 2. Parallel computation within CPU:

2. 2. 1. **find out your CPU**

2. 2. 2. computation power

2. 2. 3. Inter-computation unit connection

2. 3. 64-bit computing

2. 3. 1. have you installed a 64bit Operation system

2. 3. 2. integer type in programming language

2. 3. 2. 1. integer range

2. 3. 2. 2. Unsigned integer

2. 3. 2. 3. deal with algorithm overflow

2. 3. 3. float point or decimal

2. 3. 3. 1. sequence of float operation is important for precision

2. 3. 4. floating point overflow

2. 3. 4. 1. float point arithmetic exception

2. 3. 5. Migrate to 64bit computation

2. 4. Super computer architecture

2. 4. 1. Physical arrangement of super computer

2. 4. 2. inter-node networking

2. 4. 3. Institutional HPC

2. 5. GPU

2. 5. 1. GPU: super-computer in one die

2. 5. 2. **find out your GPU**

2. 5. 3. CPU-GPU communication

2. 5. 4. supported hardware acceleration

2. 6. Novel computation architectures

2. 6. 1. TPU for deep learning

2. 7. Quantum Computation

2. 7. 1. Qubit vs bit

2. 7. 1. 1. Qubit gate

2. 7. 1. 2. Major players

2. 7. 1. 3. Quantum computer Programming

2. 7. 2. applications
2. 8. Memory technology and CPU-thirsty
2. 8. 1. lagging memory speed
2. 8. 2. DDR technology
2. 8. 3. Multiple-channel technology
2. 8. 4. GDDR
2. 8. 5. CPU cache
2. 8. 6. Memory access and allocation is expensive
2. 9. IO Speed
2. 9. 1. PCIe IO bus
2. 9. 2. IO speed limited by system call
2. 10. Scheduling supercomputer
2. 10. 1. Software compiled for a specific HPC
2. 10. 2. Resource management
2. 10. 3. Job submission
3. Infrastructure for big data
3. 1. big data in research and engineering
3. 2. Data in memory (Volatile)
3. 2. 1. VM and program memory layout
3. 2. 2. create a large object in memory
3. 2. 3. data structure for big data
3. 3. Storage devices
3. 3. 1. External non-volatile storage
3. 3. 2. Storage tech and speed
3. 3. 2. 1. Storage hardware interface for PC
3. 3. 3. Storage array RAID
3. 4. File systems
3. 4. 1. OS-independent file path
3. 4. 1. 1. File path seperator
3. 4. 1. 2. URI/URL
3. 4. 2. Local file system types
3. 4. 3. Local storage area network (SAN)
3. 4. 4. Distributed file system
3. 4. 4. 1. Network file system (NFS)
3. 4. 5. cloud storage
3. 4. 6. Distributed/clustered FS: Storage for Big data
3. 4. 7. Parallel Distributed FS: Storage server for super computer
3. 5. Cross platform data file format
3. 5. 1. Textual file and encoding
3. 5. 1. 1. Documentation
3. 5. 1. 2. line endings in text files
3. 5. 2. Configuration file formats
3. 5. 3. binary file format and byte-order endianness
3. 6. File format for large dataset
3. 6. 1. HDF5
3. 6. 2. NetCDF v4
3. 6. 3. XDMF
3. 6. 4. XML partitioned VTK file
3. 6. 5. Parallel FileSystem and MPI-IO
3. 7. Database for big data application
3. 7. 1. Conventional RDB
3. 7. 2. In-memory DB
3. 7. 3. SQLite the single-file database
4. Parallel Programming

4. 1. OS support for parallel computation
4. 1. 1. Process and Thread management
4. 1. 2. IPC and locking
4. 1. 3. Parallel IO
4. 2. Parallel modes
4. 2. 1. Multi-threading and thread pool
4. 2. 1. 1. OpenMP
4. 2. 1. 2. OpenACC
4. 2. 2. Task Parallel with thread pool
4. 2. 3. MPI for distributed computation
4. 2. 3. 1. NUMA ()
4. 2. 4. GPU computation
4. 2. 4. 1. OpenCL and CUDA
4. 2. 4. 2. ROCm for AMDGPU
4. 2. 4. 3. sourceIntel OneAPI for CPU, GPU, etc
4. 2. 4. 4. Sycl single source GPU and CPU computation
4. 2. 4. 5. OpenMP Offload to GPU
4. 2. 5. Computation libraries support both GPU and CPU
4. 2. 5. 1. Eigen
4. 2. 5. 2. PETSc
4. 2. 5. 3. Thrust
4. 3. Factors for parallel efficiency
4. 3. 1. Scalable memory allocator
4. 3. 2. GPU and CPU data exchange
4. 3. 3. Thread creation and destroy
4. 3. 4. Comparison and selection
4. 4. Thread vs. process level parallel
4. 4. 1. Mutlithreading with shared memory address
4. 4. 2.
4. 4. 3. Workers cluster
4. 5. Race condition and synchronisation
4. 5. 1. atomics
4. 5. 2. lock, mutex, etc
4. 5. 3. Asynchronous programming
4. 6. Parallel algorithm and concurrent data structure
4. 6. 1. Parallel STL based on multi-threading
4. 6. 2. Concurrent queue and map
4. 6. 2. 1. HPX distributive MPI data structure.
4. 6. 2. 2. TBB graph, boost graph, tensorflow graph\pagebreak

5. Problem Scale-up and Break-down

5. 1. Estimate problem scale and computational complexity
5. 1. 1. Dimension explosion
5. 1. 2. Computational complexity
5. 1. 3. Resource limitations
5. 1. 3. 1. N-dimension array
5. 1. 3. 2. Sparse Matrix For large scale
5. 2. Strategy for parallel computation
5. 2. 1. segment big data
5. 2. 2. partitioning a large scale problem
5. 2. 3. geomtry decomposation
5. 3. Divide and conquer
5. 4. Communication and collaboration
5. 4. 1. introduction to IPC
5. 4. 2. Message Passing Interface (MPI)
5. 4. 3. DDS
5. 4. 4. Kafka, RabbitMQ, AMQP, Mqtt, JMS

6. The Jungle of Programming Languages

6. 1. Introduction

6. 1. 1. Timeline of programming languages

6. 1. 2. How to select and learn your language

6. 2. Compiling languages

6. 2. 1. Fortran:

6. 2. 2. C/C++

6. 2. 2. 1. Introduction to C++

6. 2. 2. 2. Evolution of C++

6. 2. 3. Compiling process of C/C++

6. 2. 4. The design of LLVM and GCC

6. 2. 5. Java, JVM and JIT

6. 2. 6. C# and dotnet framework

6. 2. 7. Other compiling languages

6. 3. Interpreting lang

6. 3. 1. Introduction

6. 3. 2. Python:

6. 3. 3. Tcl/TK: science

6. 4. Languages for computation

6. 4. 1. Matlab and similar

6. 4. 2. Other R, Julia, etc

6. 4. 3.

7. Good practice to design large C++ Software

7. 1. Don't reinvent the wheel

7. 1. 1. understandable: design patterns

7. 1. 2. Awesome C++ libraries list

7. 2. Have you really mastered C++

7. 2. 1. function signature overloading

7. 2. 2. keyword `static` and `using`

7. 2. 3. Implicit conversion

7. 3. use modern C++11

7. 3. 1. multi-threading

7. 3. 2. smart pointers

7. 3. 2. 1. new smart pointer types

7. 3. 2. 2. avoid using reference by `shared_ptr`

7. 3. 2. 3. Return only value type of smart pointers

7. 3. 2. 4. pass shared smart pointers as function parameter

7. 3. 2. 5. be careful to common errors using smart ponters

7. 3. 2. 6. `make_shared(T)` or `shared_ptr<T>(new T())`

7. 3. 2. 7. thread safety of smart pointers

7. 3. 2. 8. STL iterator is pointer `typedef`

7. 3. 2. 9. `std::any` as a better `std::shared_ptr<void>`

7. 3. 3. `std::function` and functional programming

7. 3. 3. 1. lambda function

7. 3. 4. compiling time computation

7. 3. 4. 1. constexpr

7. 3. 4. 2. Type traits and template enhancement

7. 4. C++20 and beyond

7. 4. 1. concepts

7. 4. 2. module instead of header files

7. 4. 3. library enhancement parallel

7. 4. 4. asynchronous programming

7. 4. 5. C++2y

7. 5. extensible: modulerisation

7. 5. 1. example by KDE5 tier hierachy

7. 6. Reliable: testing
7. 6. 1. unit test, encapsuation
7. 6. 2. feature/functional test (integration test)
7. 6. 3. test coverage
7. 6. 4. physical testing /market validation
8. Efficient Python Programming
8. 1. The Power of Python
8. 1. 1. The versatile language Python
8. 1. 2. search instead of reinventing the wheel
8. 2. Fast prototyping
8. 3. Version and runtime
8. 3. 1. Is your python import the correct module?
8. 3. 2. python environment virtualization
8. 4. Documentation
8. 4. 1. versatile doxygen
8. 4. 2. Sphinx and ReST
8. 5. Build your own swiss knife kit
9. Workflow automation by shell script
9. 1. the power of batch processing
9. 1. 1. why shell script in 21 centry?
9. 1. 2. POSIX shell
9. 1. 3. other scripting languages
9. 1. 3. 1. Python
9. 2. shell scripting
9. 2. 1. learn bash script in one day
9. 2. 2. Bash for advance users
9. 2. 3. pitfalls of shell script
9. 2. 4. minimal requirement
9. 2. 5. File system operation
9. 2. 5. 1. Permissions, ownership, modification time
9. 2. 5. 2. Symbolic link
9. 2. 5. 3. Misc
10. Web programming
10. 1. WWW
10. 1. 1. data spec spec HTML5 and Client side Javascript
10. 1. 2. Server-side scripting
10. 1. 3. HTML5 and dynamic pages
10. 2. Javascript programming
10. 2. 1. Object-oriented ES6
10. 2. 2. Javascript Module
10. 2. 3. JS beyond web
10. 3. Webassebmly as web VM
10. 3. 1. Emscription (compiling C++ into webassembly)
10. 3. 2. other language target web
10. 4. Network Authentication and Encryption
10. 4. 1. OAuth
10. 4. 2. HTTPS, VPN\pagebreak
11. Mixed language programming
11. 1. Introduction why mixed
11. 1. 1. Mixing C, Fortran and C++
11. 1. 2. language wrapping
11. 1. 3. dotnet CLR and Java VM

11.1.4. language independent interface
11.1.5. Debugging python module written in C++
11.2. Language binding for Python
11.2.1. write python module in C or C++
11.2.2. Boost.python and PyBind11
11.2.2.1. Cython: write python module in C++
11.2.3. cppyy: JIT and binding generation
11.2.4. SWIG for python
11.2.5. Fortran to Python
11.2.6. Qt and GTK's own wrapping
12. Architecture for Cross-platform Software
12.1. Key requirement
12.1.1. Key requirement of scientific software
12.2. Cross-platform software design
12.2.1. Cross-platform
12.2.2. Software and hardware platforms
12.2.2.1. Software platform (c++ example):
12.2.2.2. hardware platform
12.3. Windows application architecture (API)
12.3.1. The history of Windows API and application architectures
12.3.2. Facts about UWP:
12.3.3. UCRT and impact on deployment
12.3.3.1. Windows 10 SDK
12.4. Linux distro
12.4.1. Linux Fragmentation and LSB
12.5. Detect OS
12.5.1. Preprocesor for C and C++ compilers
12.5.1.1. Is there any library to do that?
12.5.2. cross-platform building system
12.5.3. cloud computation
12.5.4. challenging of testing
12.6. Sustainable Component selection
12.6.1. lifecycle plan
12.6.2. Key components
12.6.3. Tools selectoin
12.7. API design
12.7.1. Principles
12.7.2. Consistent naming convention
12.7.3. function design
12.7.4. class API design
12.7.5. API document
12.8. ABI and API compatibility
12.8.1. binary compatible is crucial for enterprise platforms
12.8.2. Compiler linkage: static or shared?
12.8.3. Find the correct shared library
12.8.4. libraries version control
12.8.5. ABI and forward compatibility
12.8.6. C/C++runtime
12.8.7. C++ pImpl Idiom for stable ABI
12.8.8. API stability
12.9. Modular design
12.9.1. module (java package) level encapsulation
12.9.2. binary plugin design
12.9.2.1. portable binary plugin system
12.10. Extensible architecture

12.10.1. Source code level extension
12.10.2. ABI level extension
12.10.3. Protocol based extensible framewrok
12.11. Accessible User interface
12.11.1. TUI, Scripting and GUI
12.11.2. Web UI and Restful API in clound compuution
12.11.3. Voice command and
12.11.4. Human-brain VR AI
13. Large Software Project Management
13.1. ## Software engineering
13.1.1. Software engineering project
13.1.2. Software development life cycle (SDLC)
13.1.3. Continuous integration
13.2. Proposal and funding
13.2.1. Funding source for initative
13.2.2. long term community-driving
13.3. Software license
13.3.1. Open source software license
13.3.2. Documentation license
13.3.3. The Creative Common Licenses
13.4. Community development
13.4.1. One dominant
13.4.2. elected committee
14. Software Engineering Process
14.1. Source management
14.1.1. Git for version control
14.1.2. Efficient team collaboration
14.2. Software testing
14.2.1. Unit test and coverage
14.2.2. Regression unit
14.2.3. Integration test
14.2.4. Physical/Market/User validation
14.3. Continuous integration (CI)
14.3.1. improve compiling performance
14.3.2. automated and parallel testing
14.3.3. Container for different platforms
14.4. Software productivity tools
14.4.1. Integrated Development Environment (IDE)
14.4.2. Other productivity tools
14.5. Coding convention and Code style
14.5.1. `API` design
14.5.2. Code style or smell
14.5.3. `const` exception thread-safety contract
14.5.4. code analysis tools
14.6. Documentation
14.6.1. generation from source code
14.6.2. structure
14.6.3. book, wiki and forum
15. Software Debug and Optimization
15.1. Debugging
15.1.1. Debugger
15.1.2. Debug info (symbol library)
15.1.3. Tools to discover potential bugs

15.2. Profiling/benchmarking
15.2.1. computation time and memory usage
15.2.2. profiling tools
15.2.2.1. `perf` is the modern tool
15.2.2.2. igprof
15.2.2.3. sprof
15.2.3. Benchmarking tools and methods
15.3. Optimization
15.3.1. Compiler optimisation
15.3.1.1. Linker-time optimization (LTO)
15.3.1.2. Machine code optimisation
15.3.1.3. Profile-guided optimization (PGO)
15.3.2. Manually optimization by refactoring
15.3.2.1. Code analysis by trace
15.3.3. Redesign the program by parallelization
16. Software Packaging and Release
16.1. Shared library dependency
16.1.1. What is shared object
16.1.2. POSIX `LD_LIBRARY_PATH`
16.1.2.1. RPATH
16.1.3. MacOX is quite different in RPATH
16.1.4. Windows DLL loading order
16.1.5. Difference between Windows and POSIX
16.2. Packaging
16.2.1. Package/installer formats on Windows
16.2.2. Packaging on Linux
16.2.2.1. Linux package formats
16.2.2.2. AppImage, snap and flatpak
16.2.3. Packaging and Package management for HPC
16.2.4. Docker image and cloud computation
16.3. Distribution channel
16.3.1. Linux official central repository
16.3.1.1. official repo
16.3.1.2. third-party repository
16.3.2. Windows software distribution
16.3.2.1. Application distribution
16.3.3. Cloud deployment (no installation needed)
16.4. Payment
16.4.1. One-off payment
16.4.2. Annual subscription
16.4.3. Open source RedHat mode
16.4.4. Non-compulsary payment: donation
16.5. License enforcement
16.5.1. Open source requirement
16.5.2. Network license manager
16.5.3. source code protection
16.6. Post-release: Bug tracking
16.6.1. Predictable and frequent release
17. Refactor legacy project
17.1. Reason, tools for refactoring
17.1.1. why needed
17.1.2. process
17.1.3. tools for refactoring
17.2. Porting to another platform

17.3. Redesig and rewrite
18. Computational Mathamatics
18.1. linear algebra
18.1.1. **Basic Linear Algebra Subprograms**(BLAS)
18.1.2. LINPACK and LAPACK benchmarking
18.1.3. PETSc with GPU and MPI
18.2. Numerical method ODE and PDE
18.3. Computational geometry
18.3.1. Computational geometry kernels
18.3.2. Open source libraries
18.3.3. OpenCASCADE
18.4. Topology and Graph theory
18.4.1. networkX for python
18.4.2. `boost::graph` for C++
18.5. statistics and probability
18.6. stochastic methods
18.6.1. monta-carlo methods
18.7. misc
18.7.1. Symbolic math
18.7.2. cryptography and informatics security
18.7.3. bitcoin, chainzone
19. Computional Physics software
19.1. Methods, spatial and temporal scale
19.2. Macroscale simulation
19.2.1. Plasma physics: Bout++, CFD
19.3. Mesoscale simulation
19.3.1. LBM
19.3.2. Monta-Carlo
19.4. Molecular dynamics
19.4.1. Lammps
19.5. Quantum mechanics
20. Open source Computer-aided engineering (CAE)
20.1. Design: CAD
20.1.1. CAD kernels
20.1.2. Open source CAD
20.1.3. Data exchange
20.1.3.1. + STEP 242
20.2. Domain Partitioning
20.2.1. ParMETIS [1]
20.2.2. SCOTCH and PT-SCOTCH [1]
20.2.3. Hypre
20.3. Preprocessing: Meshing
20.3.1. Meshing methods and file formats
20.3.2. Netgen and GMSH
20.3.3. SALOME and smesh
20.4. Simulation: FEA
20.4.1. Dolfin (FEniCS)
20.5. Simulation: CFD
20.5.1. OpenFOAM
20.6. Post-processing: Visualization
20.6.1. The design of Paraview

20.6.2. OSPRay
20.7. Post-processing: Optimization
20.8. Automated CAD to CAE pipeline
20.9. Misc
20.9.1. Dimension analysis, units
21. Industrial Internet of Things (IIoT)
21.1. Big picture
21.1.1. SCADA architecture
21.1.2. Device Communication infrastructure and methods
21.1.3. Communication realtime DDS
21.2. OPC-UA / IEC 62541
21.2.1. Introduction to OPC-UA
21.2.2. OPC-UA open source implementation
21.2.3. tutorial on FreeOpcUa
21.2.3.1. Design
21.2.3.2. installation
21.2.3.3. start server and then client
21.3. RTOS and Embedded System
21.3.1. RTOS
21.3.2. GUI, Networking, Security, IO
21.3.2.1. IO (I2C, SPI, UART, CAN) to connect to sensor
21.3.2.2. Networking: mobile, Ethernet, wifi, Laora,
21.3.3. Middle-ware
21.3.4. Applications of Embeded system
21.3.5. Testing and Debugging RTOS
22. Big data and AI
22.1. TensorFlow for AI
22.1.1. Design of operator and executor
22.1.2. Language binding
22.2. Data science
22.3. Business intelligence
23. Closing Remarks