Parallel Design Patterns-L07

Parallel Frameworks I

Course Organiser: Dr Nick Brown
nick.brown@ed.ac.uk
Bayes room 2.13

Introduction to frameworks

- Parallel Frameworks
  - Why?
- What makes a good parallel framework?
- Common parallel examples as an illustration of frameworks
  - Common frameworks used in HPC codes

Parallel Frameworks – Why?

- Lots of people write code
- Quite a lot of people write parallel code
- Few people write good parallel code
- Many parallelisations follow similar patterns
- Is it possible to separate the parallelism from the details of the problem?
  - Not entirely, as you will have already seen and will continue to see in this course
  - Things can be done. There is progress to be made…
  - Not everyone agrees that frameworks are the way to address this problem
- Why reinvent the wheel?

Who will write the parallel frameworks?

- People like you?
- Frameworks have to bridge the gap between applications and the underlying architectures
- Need to abstract away unimportant details of each one from the other *while maintaining as much knowledge as possible about available parallelism*
  - This isn't about writing serial code and getting the compiler/machine to parallelise it; it's about separating out the details of what can be done in parallel from how it is done
  - Continued work required to adapt frameworks for new architectures
    - with the aim of maintaining a stable API
  - Higher level programmers (e.g. scientists) will probably still have to learn to "think parallel"

Software frameworks

- An abstraction where software provides some generic functionality that can be used or modified by additional user written code that results in application specific software.

- Reusable semi finished architectures

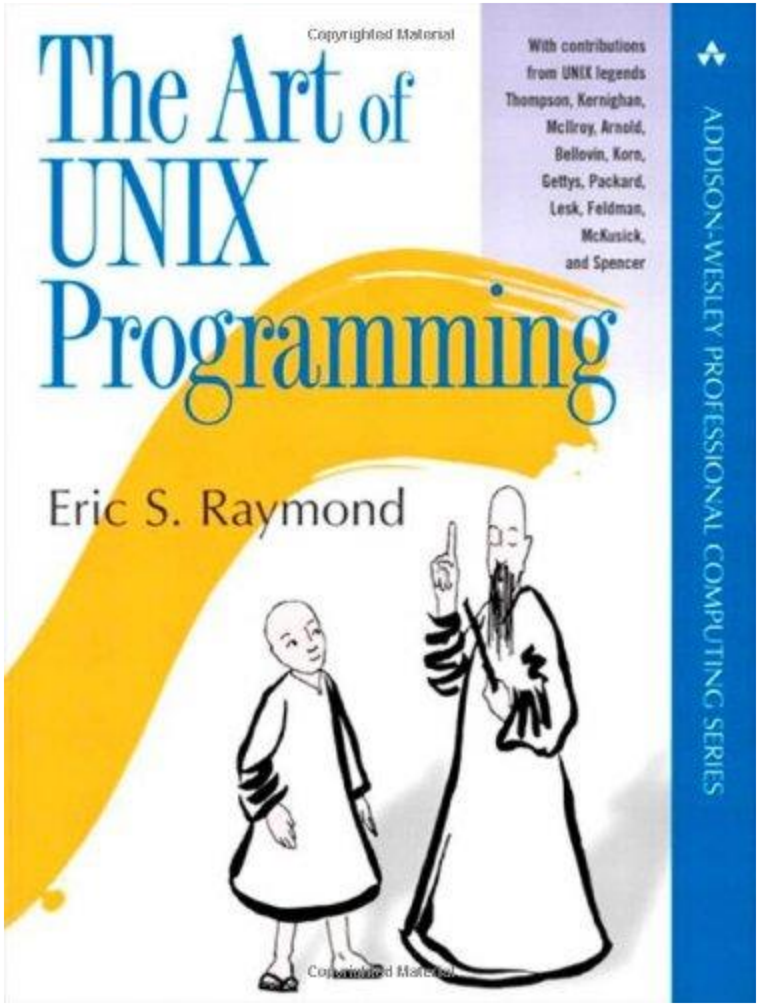
- E.g. Some combinations of
  - Libraries / APIs
  - Middleware / runtimes
  - Standard ways of expressing parallelism such as skeletons

Software frameworks vs libraries

- The line can be blurred between frameworks and libraries. Frameworks typically manage an aspect of your code (e.g. the parallelism, IO, computation etc..)
- To differentiate consider
  - **Inversion of control**: A framework often controls the program flow in some or all of the application
  - **Extensibility**: A framework should be extensible by the user by them providing some code
  - **Default behaviour**: Sensible defaults are often present if the user does not wish to override this
  - **Immutability**: The framework code itself should remain unchanged but can be extended via the user
  - **Data types:** Frameworks often define their own types which are used in your code
  - **Determine key aspects:** Such as data decomposition

Nine rules for writing parallel codes….

Book



- Written by one of the key people in the UNIX scene
- Whilst this book is about UNIX, the lessons can equally be applied to software more generally
- HPC and UNIX share many similar traits.

http://catb.org/esr/writings/taoup/html

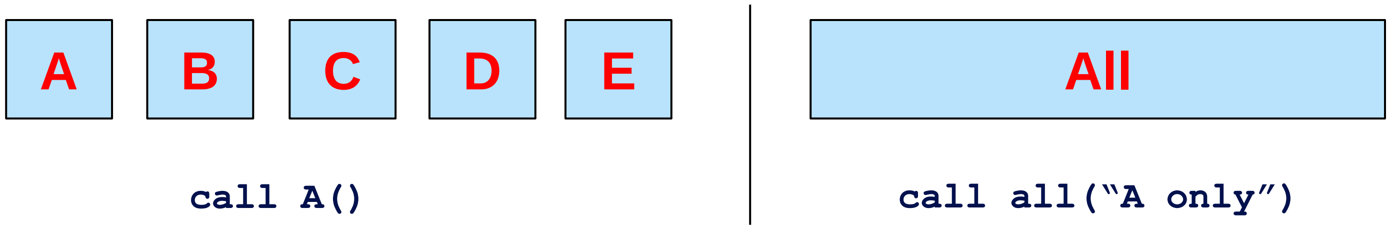
Raymond's Seventeen Rules

- Rule of Modularity
- Rule of Clarity
- Rule of Composition
- Rule of Separation
- Rule of Simplicity
- Rule of Parsimony
- Rule of Transparency
- Rule of Robustness
- Rule of Representation
- Rule of Least Surprise
- Rule of Silence
- Rule of Repair
- Rule of Economy
- Rule of Generation
- Rule of Optimisation
- Rule of Diversity
- Rule of Extensibility

Rule of Modularity

- Write simple parts connected by clean interfaces
- Holds the global complexity down

- Users of your framework can very easily select which bits to use and which bits not to use
  - As opposed to some obscure argument rules in the API which determine this.
  - In HPC want to have a very clear idea of what is running



- Can also be a way in which users can extend your framework (which we will see in an example later)

Rule of Separation

- Separate policy from mechanism
- Policy
  - Provided by the user (they know what they want best)
- Mechanism
  - Provided by your framework.
  - Often the tricky, crucially important, low level and uninteresting (to a user) aspects such as highly efficient halo swapping.
- Sometimes need careful thought how to inject the policy into the mechanism

Rule of Simplicity

- Design for simplicity, add complexity only where you need it
  - Best type of optimisation is not to write it in the first place!

- Pressure to make codes more complex
  - Technical machismo
  - Changing requirements

- Complexity results in bloated, huge buggy, brittle codes

- Actively resist bloat and complexity
  - Small is beautiful
  - Need to accept that there is a high value on simple solutions

Rule of Transparency

- Design for visibility
  - Transparency is when you can look at something and immediately understand what is going on
  - Discoverability is when, with the provided tools, you can investigate something and understand what is going on
- E.g are API calls orthogonal or are there many flags and mode bits which means that a single call might do multiple tasks?
- Is it easy to find the appropriate part of the code called by a specific function?
- What about documentation
  - Tools such as Doxygen

Rule of Robustness

- The child of transparency and simplicity
- Users of a framework might very well use it in ways that the developers have not anticipated
  - But if you say that if something does “X” then it should do “X”
- Unusual parameters
  - Such as extra long or empty strings
  - Negative numbers
- How can you convince potential users that your framework is robust?
  - Good practice such as documentation, unit testing, easy building goes a long way
  - A well thought out design and API

Rule of Least Surprise

- Your framework should always do the least surprising thing
  - Demands the least amount of learning from the users
  - Most effectively connect to user’s pre-existing knowledge

- What is the likely background of your users, are there conventions that you can adopt?
  - For instance from some existing packages they use?

- More general technical tradition
  - E.g. Climate and Forecast (CF) convention for data files
  - Format of configuration files (e.g. FORTRAN NAMELIST)
  - Common command line switches
  - Programming language/implementation technologies

Rule of Silence

- When your framework has nothing surprising to say, then keep quiet
- User attention is a precious resource
- If you want to have debugging information then have this as a flag
  - Ideally via a preprocessor directive so your code is not checking a conditional at runtime

Rule of Repair

- Repair what you can, but when you must fail noisily and as soon as possible
- If software can adapt to unexpected conditions then great
  - But some of the worst kind of bugs are when a repair fails and the problem quietly causes some corruption.
- When you must fail, then fail transparently
  - Such as generating a core dump

Rule of Extensibility

- Design for the future, because it will be here sooner than you think.

- How extensible is your API?
  - Or would it need to be deprecated?
- Can users easily extend your framework
  - Can new functionality be easily plugged in?
  - Make the joints in your framework flexible
- How extensible are your data formats?
  - Including a version number can go a long way here
  - Self describing data formats can be very useful
- How about machine specific optimisation?

Focusing more on HPC

Focussing in on HPC

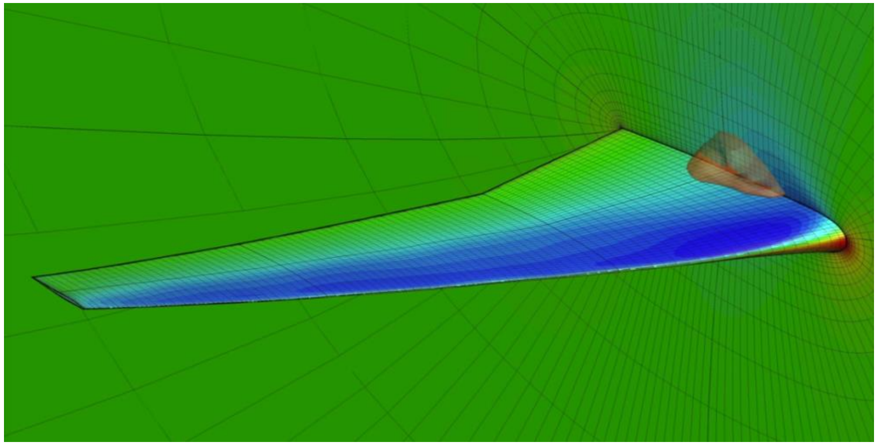
- These rules are based on Raymond's UNIX experience
- But HPC is a far smaller community with fewer resources
  - People are often working individually on codes
  - Driven by science, frameworks often start life unplanned and grow organically
  - Do you want to be inundated with lots of requests for support after funding has expired?
  - Often a developer/user's task to install required frameworks
- Scalability and performance are crucial
  - Raymond argues that programmer time is more precious than machine time, but in HPC we don't have this luxury
  - In our field performance portability is very important

What makes a good parallel framework?

- There are a number of key objectives
  - **Integration:** With existing codes or codes that do a variety of different functions. Also integration with the community.
  - **Visibility:** Is this framework going to impact upon performance or scalability? Is it bit reproducible & does this matter?
  - **Documentation:** Clear instructions (and ideally examples) about how to use this?
  - **Flexible:** Can I do what I need to using this?
  - **Portable and supported:** Already installed and configured on the systems you plan to use?
  - **Extensible:** Can the framework be extended as your code evolves?

PETSc

- Portable Extensible Toolkit for Scientific computation
- Used for solving systems of differential equations (such as Laplace’s equation from the practical.)
- Geometric decomposition



- Problems plug into the framework, can manage parallelism (if you wish), plenty of different choices such as the solvers and preconditioners to use can be selected at runtime

Using PETSc

We have already initialised PETSc which will, amongst other things, create its options database

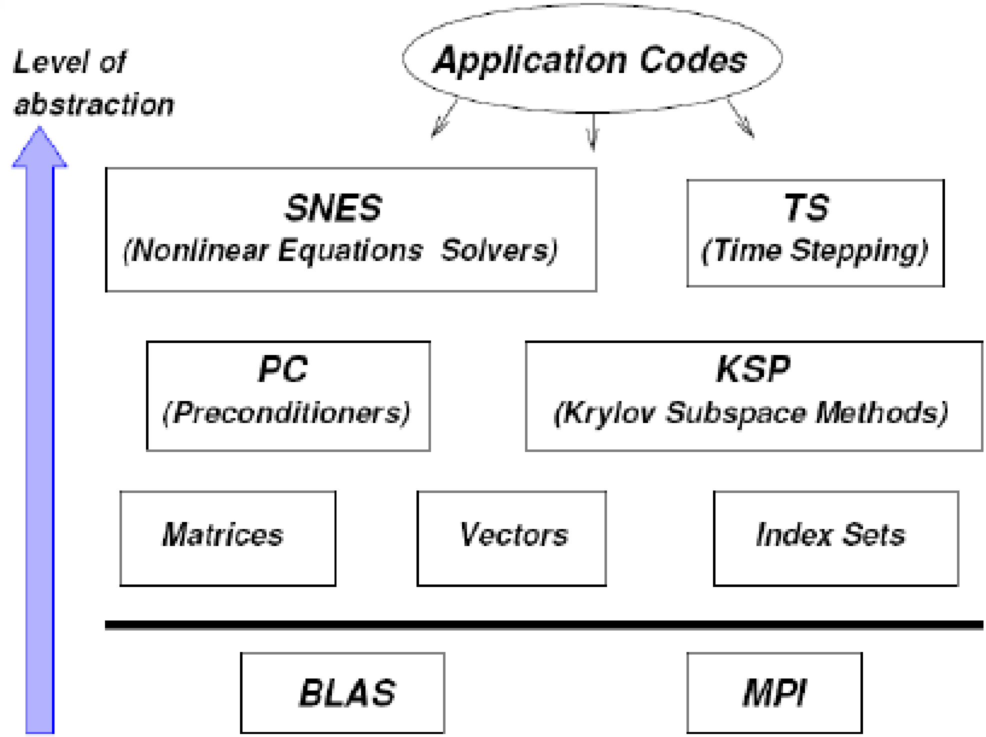
```
KSP ksp
DM da
Vec x
integer ierr

call KSPCreate(PETSC_COMM_WORLD,ksp,ierr)
call KSPSetFromOptions(ksp,ierr)
call DMDACreate3d(PETSC_COMM_WORLD,DMDA_BOUNDARY_PERIODIC,….,nx,&
        ny,nz,npx,npy,npz,…, da,…)

call DMSetInitialGuess(da,ComputeInitialGuess,ierr)
call KSPSetComputeRHS(ksp,ComputeRHS,PETSC_NULL_OBJECT,ierr)
call KSPSetComputeOperators(ksp,ComputeMatrix,PETSC_NULL_OBJECT,ierr)

call KSPSetDM(ksp,da,ierr)
call KSPSolve(ksp,PETSC_NULL_OBJECT,PETSC_NULL_OBJECT,ierr)

call KSPGetSolution(ksp,x,ierr)
```

Using PETSc

- Has a beginner, intermediate, advanced and developer API
  - The idea being that it can be understood by a variety of users
  - It is well documented and with plenty of examples
- Supports multiple problem abstractions
  - The scientist's view of Ax=b, they provide functions which build up the matrix A and the initial conditions in a vector b. The output from this is the vector x. Using the DMDA calls these can be automatically partitioned and distributed amongst the UEs.
  - Alternatively the programmer can supply routines that define the behaviour which is more advanced but saves explicitly building the matrix
- Numerous third party additions have been written which add additional functionality to the framework

PETSc organisation

- Operations are defined upon matricies, vectors and index sets, these contain all the parallelism.
  - Sequential matrix and vector results in sequential computation
  - MPI matrix and vector result in MPI parallelisation
  - Threaded matrix and vector result in threaded parallelisation



- In order to support a new architecture (i.e. GPUs or hybrid MPI-OpenMP) then just need to define a new matrix and vector, with the rest of the framework remaining unchanged.

PETSc

- Facets:
  - **Integration:** Yes, but the temptation can be to rely on the PETSc data structures (such as Mat and Vec)
  - **Visibility:** The solver and preconditioners are well known, published benchmarks for DMDA and is open source.
  - **Flexible:** Lots of options
  - **Portable and supported:** Commonly installed and trivial to build (but does take a long time!) Email support lists
  - **Extensible:** Hierarchical abstraction and organisation
  - **Documentation:** Extensive online documentation and examples. Differentiated for different users and examples
- Disadvantages
  - API can change dramatically between minor releases i.e. 3.5.1 -> 3.5.2 and no backwards compatibility
  - It is possible, but a bit more difficult, to do parallelism without DMDA
  - You might pay a small computation price

FFTW

- The Fastest Fourier Transformation in the West
- Computing the discrete Fourier transform is a very common operation, hence FFT algorithms are very common

e.g. noise removal

"the most important numerical algorithm in our lifetime." Gilbert Strang, Linear Algebra and Its Applications

Using FFTW

- The programmer tells the framework specifics about their problem and this then produces a plan
  - This is (potentially) an expensive operation, so you only want to do it once
  - FFTs are often called in some sort of timestep so you want the actual computation call to be as fast as possible
  - Some FFT kernels also require even data of n^2, if your data does not look like this then FFT can handle it (or select a different kernel.)

- The programmer then calls FFTW with their plan which will then do the actual FFT.

Using FFTW

```
ptrdiff_t N0=n, N1=n

ptrdiff_t alloc_local, local_n0, local_0_start;

alloc_local = fftw_mpi_local_size_2d(N0, N1, MPI_COMM_WORLD, &local_n0,
&local_0_start);

fftw_complex * data = fftw_alloc_complex(alloc_local);

fftw_plan plan = fftw_mpi_plan_dft_2d(N0, N1, data, data, MPI_COMM_WORLD,
         FFTW_FORWARD, FFTW_ESTIMATE);

... Initialise data to some values ....

fftw_execute(plan);
fftw_destroy_plan(plan);
```

Evaluation of FFTW

- Facets:
  - **Integration:** Yes, easy to limit calls to a specific function. Type definitions are simple and well documented
  - **Visibility:** Benchmarks and source code available for the library
  - **Flexible:** Lots of options and interfaces available
  - **Portable and supported:** Very commonly installed on machines and trivial to compile
  - **Extensible:** Not so much as PETSc, but source code is available
  - **Documentation:** Well documented, lots of examples. API split into beginner, advanced and guru sections.
- Disadvantages
  - It is commonly accepted that the computation aspects are good but the parallel (communication) aspects are not. Therefore often people do their own decompositions and just call the 1D kernels
  - Only slab decomposition in 3D supported, although third party libraries have been developed to address this

NetCDF

- An IO framework, defines and works with files that are self describing and this is very common in HPC scientific applications.

- The basic version is serial, but parallelism is supplied and it is trivial to move to doing parallel IO.
- Uses MPI-IO and HDF5 but the programmer is completely abstracted from the details of these.

- Comes with tools for managing and visualising the data files

Writing a NetCDF file

```
int data_out[NX][NY];
… populate data_out …..
int ncid, dimids[2], varid;

nc_create(FILENAME, NC_CLOBBER, &ncid);
nc_def_dim(ncid, "x", NX, &dimids[0]);
nc_def_dim(ncid, "y", NY, &dimids[1]);
nc_def_var(ncid, "data", NC_INT, 2, dimids, &varid);
nc_enddef(ncid);

nc_put_var_int(ncid, varid, &data_out);

nc_close(ncid);
```

Writing a parallel NetCDF file

```
int ncid, dimids[2], varid; size_t start[2], count[2];

start[0] = mpi_rank * NX/mpi_size; start[1] = 0;
count[0] = NX/mpi_size; count[1] = NY;

nc_create_par(FILENAME, NC_NETCDF4|NC_MPIIO, MPI_COMM_WORLD,
      MPI_INFO_NULL, &ncid);
nc_def_dim(ncid, "x", NX, &dimids[0]);
nc_def_dim(ncid, "y", NY, &dimids[1]);
nc_def_var(ncid, "data", NC_INT, 2, dimids, &varid);
nc_enddef(ncid);

nc_put_vara_int(ncid, varid, start, count, &data_out)
nc_close(ncid);
```

Parallel reading a NetCDF file

```
int ncid, varid, size_t start[2], count[2];


nc_open_par(FILENAME, NC_NETCDF4|NC_MPIIO|NC_NOWRITE,
      MPI_COMM_WORLD, MPI_INFO_NULL, &ncid);
nc_inq_varid(ncid, "data", &varid);
nc_get_vara_int(ncid, varid, start, count, data);


nc_close(ncid);
```

You can use other calls to inquire about the dimensions too

Evaluation of NetCDF

- Facets
  - **Integration:** Easy to integrate with existing codes
  - **Visibility:** Not particularly transparent, can be hard to get good performance
  - **Flexible:** There is plenty of documented functionality
  - **Portable and supported**: Installed on most machines, relatively easy to compile and actively developed
  - **Extensible:** Open source but not designed to be user extensible as such.
  - **Documented:** Plenty of documentation, examples and tutorials available
- Disadvantages
  - Can be hard to get good parallel performance, whilst the framework promotes abstraction you can often end up working with lower level settings

Summary

- In this lecture we have considered the design of frameworks
- Some considerations to bear in mind when designing or selecting a framework
- Many of the suggestions apply equally well to writing good HPC code, even if it is not part of a framework.

- In the next lecture we will consider some frameworks which abstract parallelism and a more concrete look at the implementation of frameworks