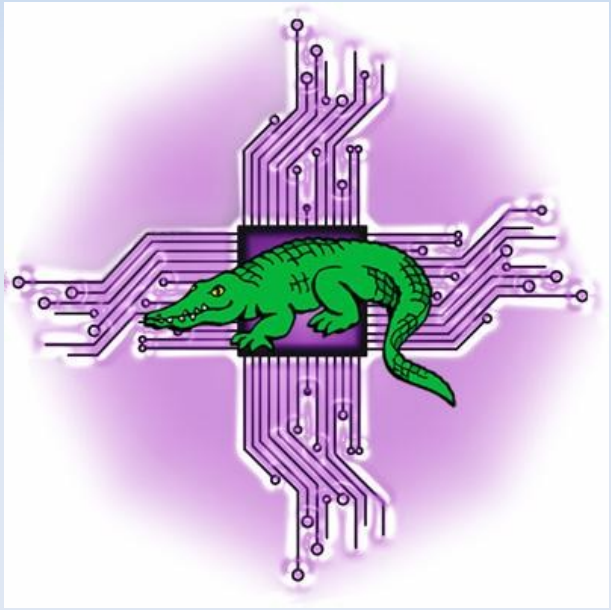
Scheduler Parallelisation

Spartan Champions Workshop



University of Melbourne, March 7, 2025

lev.lafayette@unimelb.edu.au

Outline of Lecture

This presentation and workshop will include:

The standard computing model of uni-threaded instructions and data.

Automation through looping and conditionals. Multithreaded and multi-core parallelisation.

Simple job arrays; topology issues, array ranges and variables, arrays of programs.

Scheduler parallelisation with conditions; job dependencies with job arrays.

Heredocs; using a shell scripting convention to create jobs and looping submission.

The collective noun for hedgehogs is an array.

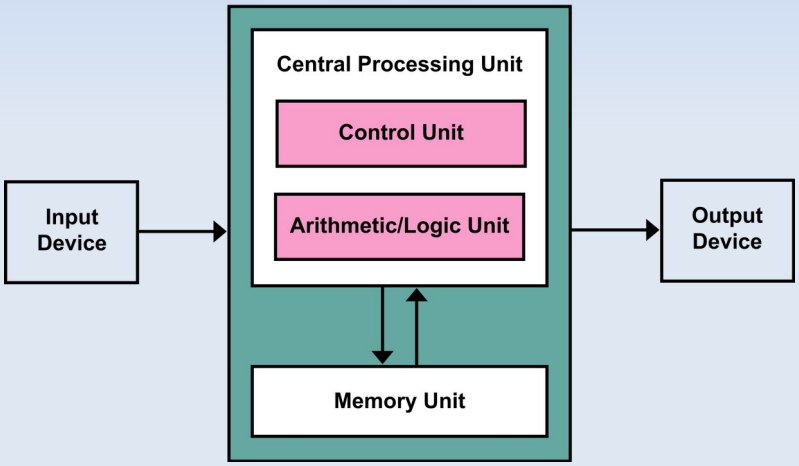
Standard Programming

A computer system (Flynn, 1966) can be thought of an instruction stream interacting with a data stream.

The standard approach assumes a single instruction stream and a single data stream (SISD).

This also is represented in physical systems as the von Neumann architecture (1954) where the instruction fetch and data operation cannot be simultaneous.

The Flynn logically can be elaborated to include single instruction to multiple data streams (SIMD), multiple instructions to a single data stream (MISD), and multiple instructions to multiple data streams (MIMD).



(img from: https://commons.wikimedia.org/w/index.php?curid=25789639)

Automation with Control Flow

A well-established means to automate procedures is through looping control flows, whether by count-control (e.g., for loops) or condition controls (e.g., do-while, repeat-until), both with stops (break, continue).

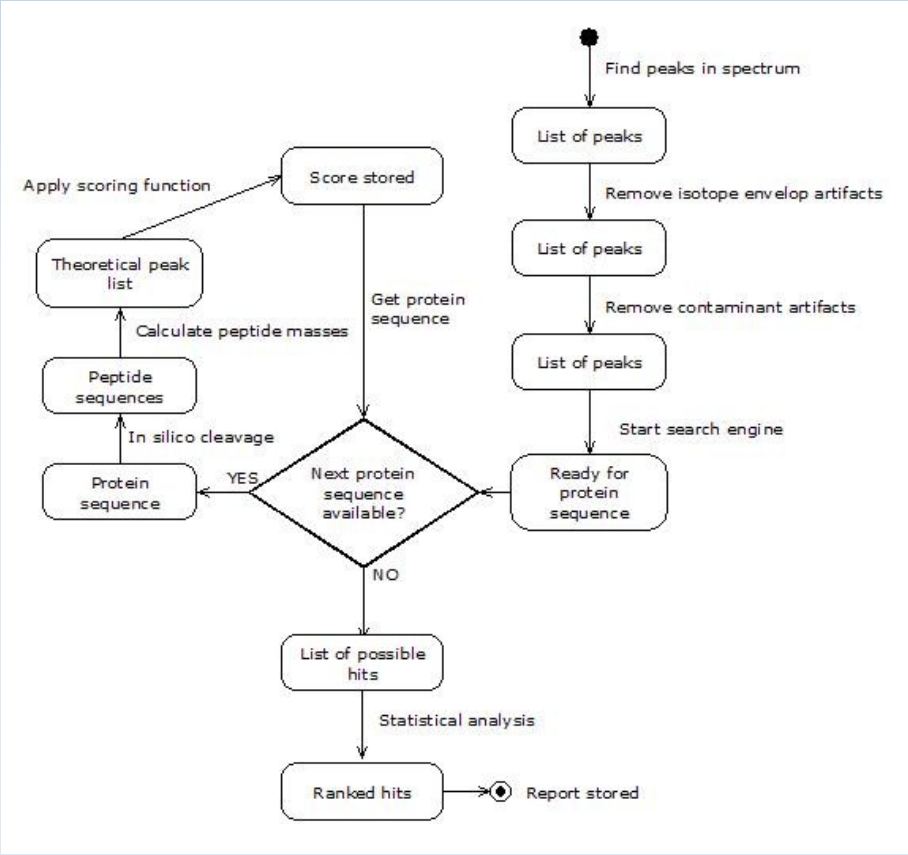
These can be elaborated further with conditional branching (e.g., if-then-else, switch).

In all cases these automate the instruction stream and are highly recommended.

However, they can be very inefficient in a multicore environment. e.g.,

for song in {1..100}; do Rscript analyse.r "song_$song.csv"; done

(Each analyse.r takes three hours to run, 300 hours in total)



Job Array Concepts

Job arrays are a directive to the scheduler to submit a job with multiple subjobs that use the same resource requests.

These are submitted simultaneously and will run independently, and therefore can be managed independently (e.g., scancel).

They also generate a bash variable which can be used to index datasets or instruction sets. The scheduler only cares about the resource requests.

They provide an easy way to parallelise and automate tasks that can generate impressive performance improvements. e.g.,

```
#SBATCH --array=0-100
Rscript analyse.r "song_$
{SLURM_ARRAY_TASK_ID}.csv"; done
```

(Each analyse.r takes three hours to run simulatenously, three hours in total)

Job Array Components

As a directive, the job array must specify indices as a range, comma-separated, or with step sizes. It must be an integer (and jobIDs are integers)! Array sizes and max index values are set in Slurm configuration (100K on Spartan).

The array can reference files according to the index value. Files without an index value can be altered with a short for loop.

An alternative is to use sed against a list of the files (or directories) to create the index reference. The array index size still must be specified in the scheduler directives!

See examples:
http://levlafayette.com/files/2025_arrays.txt

Arrays have a scheduler cost; inefficient if task is <10 minutes, use multiple job steps instead.

Job Arrays and Dependencies

Arrays can be used with job dependencies. Job dependencies set up conditional tests when a job can start (e.g., a job has run successfully, or has started, or has failed).

For example an array job can have a preceding job, such as one which creates multiple datasets.

.. or an element of an array can become the dependent job for a standard job

.. or an element of an array can be a dependency for another job element in a different array.

In all cases the same submission approach can be used.

$ sbatch --dependency=afterok:jobid1 job2.slurm

HereDocs

Heredocs allow for the construction of array-like scripts which can be launched through a loop (not *run* as a loop, *launched*).

The Heredoc establishes a block of text which is treated as a separate file.

With a loop and variables, it can be used to create multiple job submission scripts which, with branching, allows for creation of jobs with different resource requests.

Gaussian example at:
http://levlafayette.com/node/539

Individual jobs are launched, rather than subjobs, but the effect is the same.

Very handy at institutes that have blocked job arrays!


THANKS FOR WATCHING
& LISTENING PATIENTLY
makeameme.org