MANCHESTER 1824
The University of Manchester

APT

Task Scheduling for Heterogeneous Architectures

Nuno Miguel Nobre
Supervisors: Graham Riley and Antoniu Pop

MANCHESTER 1824
The University of Manchester

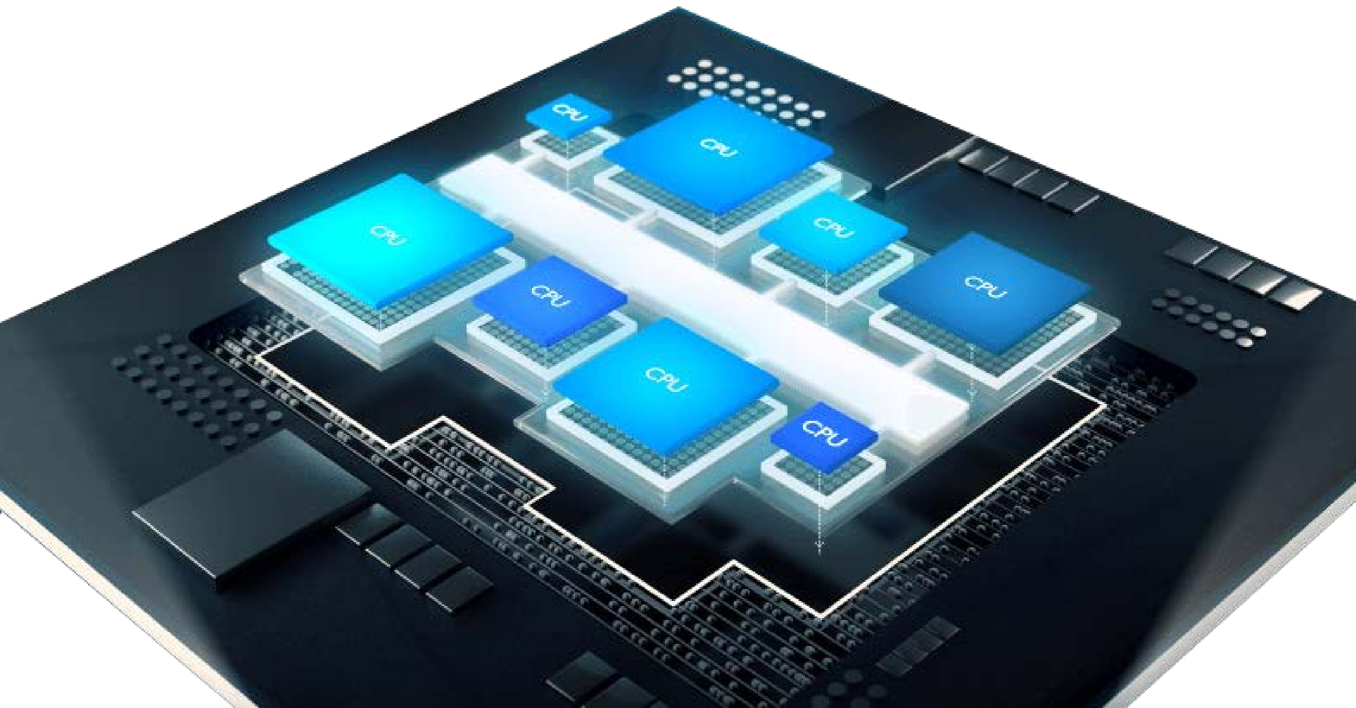
APT

Why heterogeneous computing?

Heterogeneous multi-core architectures

↓ e.g.

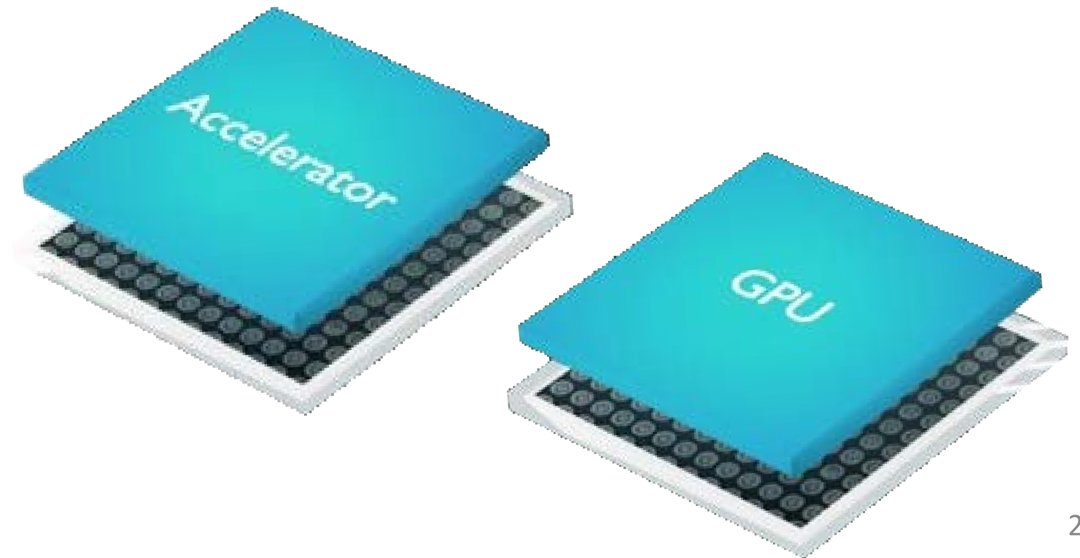
Different performance/power optimization points



Hardware accelerator-based computing

↓ e.g.

Data parallelism in GPGPUs



Task scheduling

Tasks are an application’s units of work

Scheduling involves
- Mapping tasks to each resource
- Ordering tasks on each resource
- Ensuring dependences are respected

It’s a hard problem, literally! NP-hard ⊕ crucial to leverage parallel resources

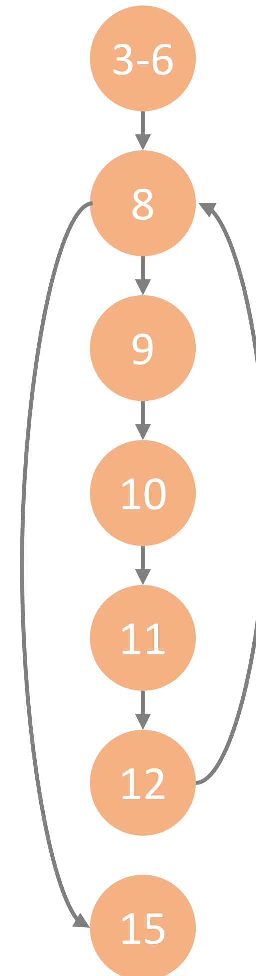
Ingredients
- Model, i.e. a way to represent tasks and their dependences
- Scheduling algorithm

Model: directed graphs

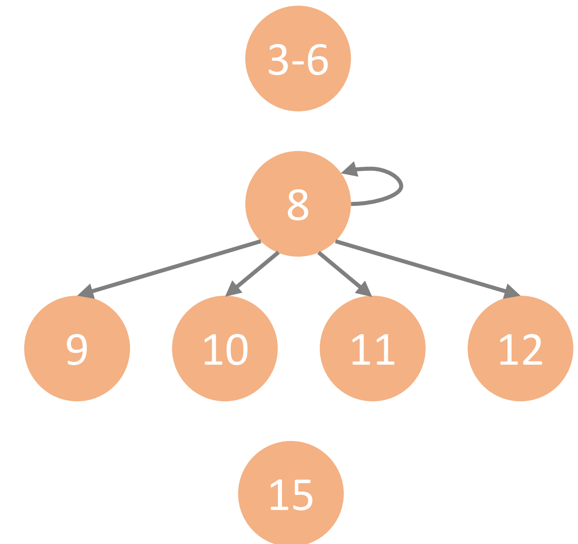
Numerical integration
C (imperative)

```
 1 double integrate(){
 2
 3   double old_y = 0.0;
 4   double x = 0.0;
 5   double y = 0.0;
 6   double s = 0.0;
 7
 8   while (x < 1.0){
 9     old_y = x * x;
10     x = x + 0.02;
11     y = x * x;
12     s = s + 0.01 * (old_y + y);
13   }
14
15   return s;
16 }
```

Control Flow Graph



Control Dependence Graph

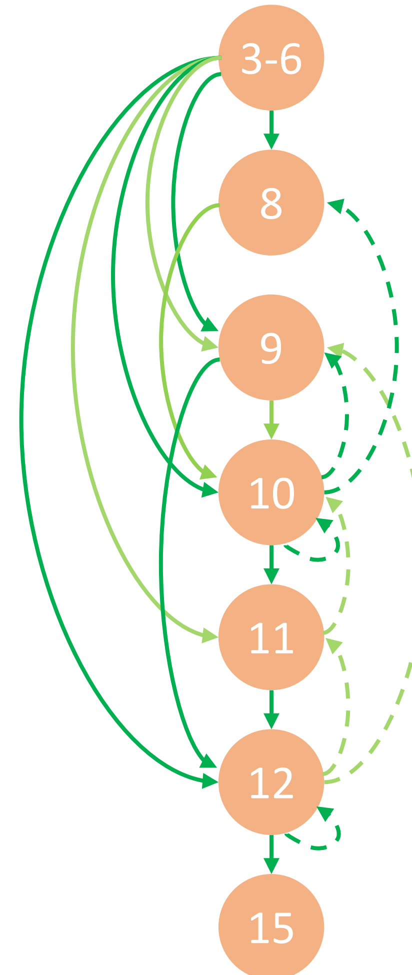


Model: directed graphs

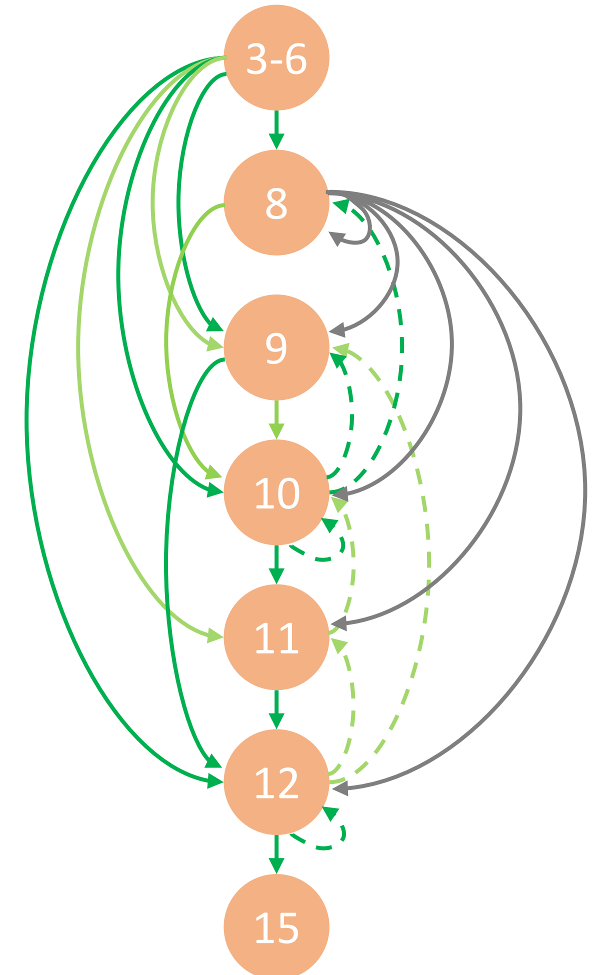
Numerical integration
C (imperative)

```
 1 double integrate(){
 2
 3   double old_y = 0.0;
 4   double x = 0.0;
 5   double y = 0.0;
 6   double s = 0.0;
 7
 8   while (x < 1.0){
 9     old_y = x * x;
10     x = x + 0.02;
11     y = x * x;
12     s = s + 0.01 * (old_y + y);
13   }
14
15   return s;
16 }
```

Data Dependence Graph



Program Dependence Graph

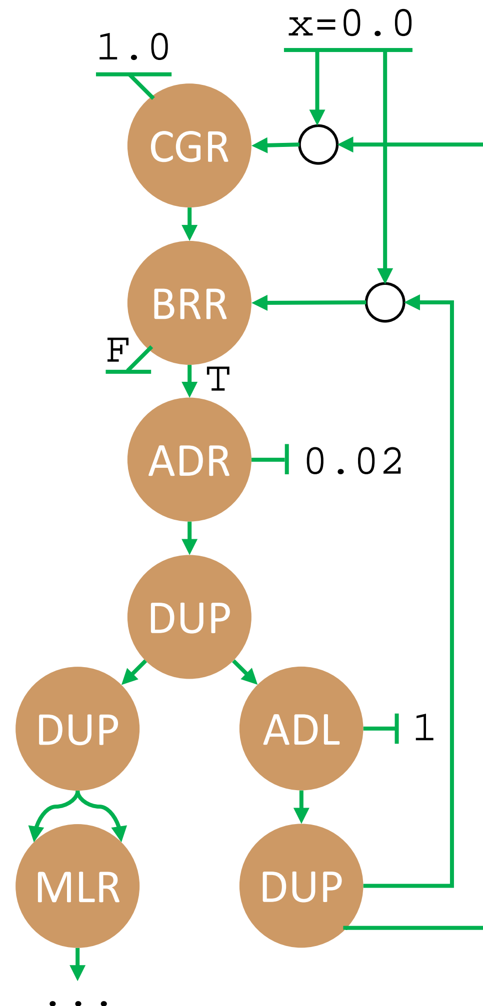


Model: directed graphs

Numerical integration
SISAL (single-assignement/functional)

```
1  function integrate(returns double_real)
2
3    for initial
4      s := 0.0;
5      y := 0.0;
6      x := 0.0;
7    while x < 1.0 repeat
8      s := 0.01 * (old y + y);
9      y := x * x;
10     x := old x + 0.02;
11   returns value of sum s
12   end for
13
14 end function
```

Data Flow Graph



Task Dependence Graph

Directed **Acyclic** Graph

Easier to { deal with reentrant code; verify feasibility }

Unified data flow approach as DFG

Unlocks asynchronous parallelism

Built by unrolling/coarser granularity

Schedulers

The DAG can be scheduled either

- **statically**, when tasks' execution times, communications costs and dependences are known
- **dynamically**, to deal with indeterminism: user input, randomness, shared resources, DVFS, …

NP-hard ⇒ heuristics: list schedulers

- an ordered list of tasks is built by assigning priorities to each task
- the highest-priority ready task is selected
- a processor is chosen that minimizes a predefined cost function
- common for their simplicity/performance

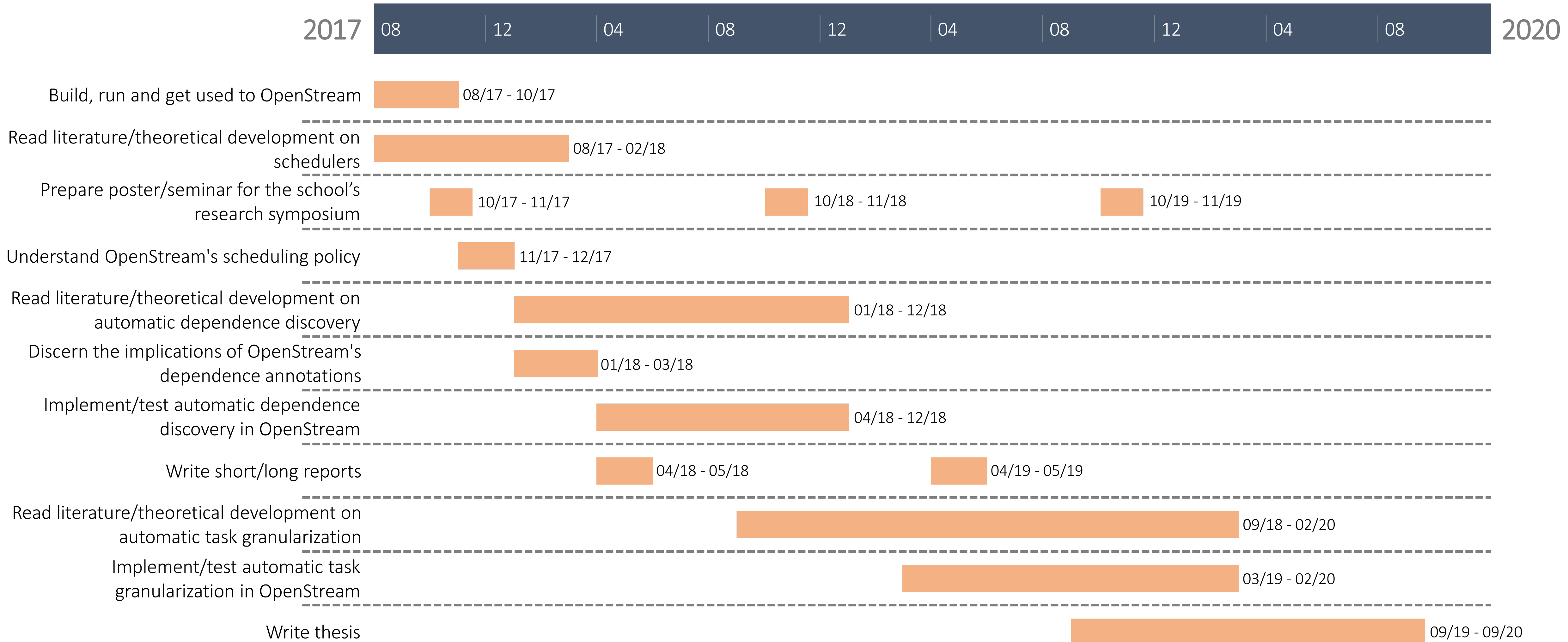
Programming models: user burdens

Programming models, like OpenStream and OmpSs, **require the user to specify the data flow between tasks and their granularity**

This not only makes **coding harder** and more **error prone**, but also leads to **suboptimal decisions**

Is it possible to achieve the same parallelization benefits that data flow programming models give users, whilst still providing intuitive language constructs that do not obfuscate the program’s purpose?

Future work



The University of Manchester

APT

Who else is working on task scheduling?

MIT IBM Cray

Cilk X10 Chapel

OpenMP ARB

OpenMP 3.0+

Intel

Threading Building Blocks

Barcelona Super-Computing Center

StarSs/OmpSs

University of Manchester

INRIA Paris

OpenStream

Dataflow computer

Polytope model

University of Tennessee

QUARK

INRIA Bordeaux

StarPU