

Task Scheduling for Parallel Systems

Oliver Sinnen

**Electrical and Computer Engineering
University of Auckland**

www.ece.auckland.ac.nz/~sinnen/
o.sinnen@auckland.ac.nz

Parallel Programming

Sequential
programming

application
specification

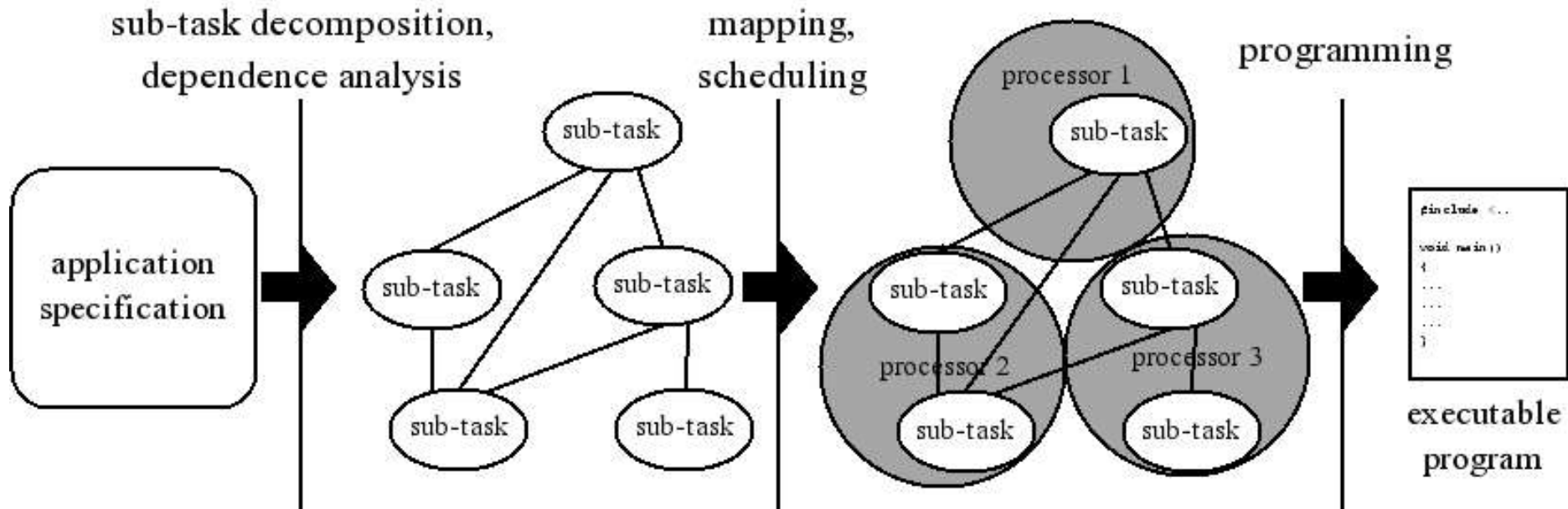
programming

```
#include <...>

void main ()
{
    ...
    ...
    ...
}
```

executable
program

Parallel Programming



Outline



O. Sinnen, “Task Scheduling for Parallel Systems”, John Wiley, 2007

- I: Introduction to task scheduling
 - List scheduling
- II: Contention scheduling
 - Awareness of communication contention in task scheduling

Current research example

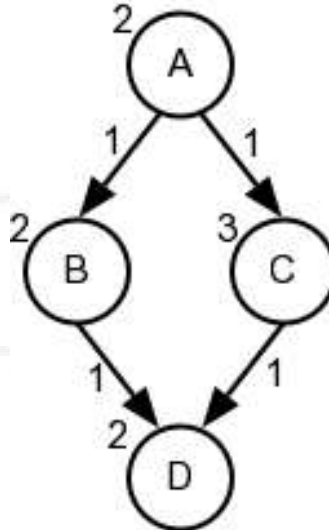
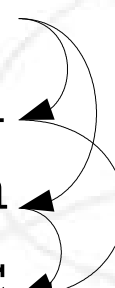
- III: Generating the Task Graph
 - Extending OpenMP

I: Introduction to task scheduling

Graph representation of program

Example: task graph (DAG)

A: $a = 1$
B: $b = a + 1$
C: $c = a * a$
D: $d = b + c$



- Graph representation of program
- Input of task scheduling

directed acyclic graph
(DAG)

node (n): sub-task

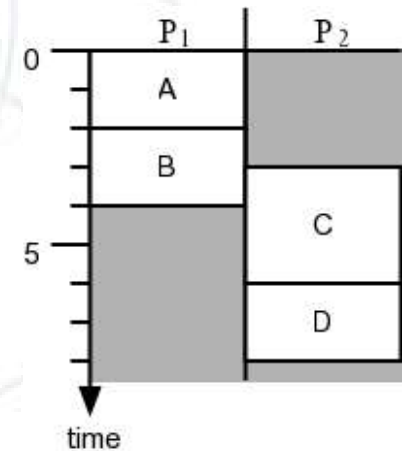
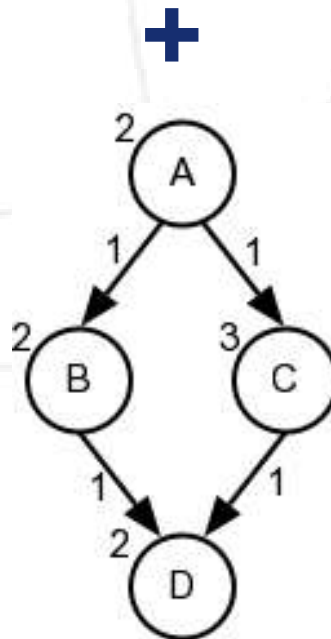
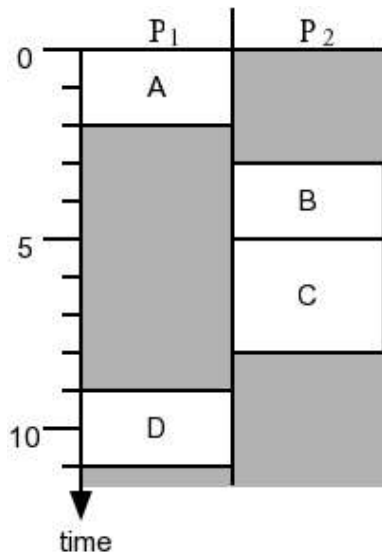
edge (e): dependence
(communication)

weight: computation $w(n)$ or
communication time $c(e)$

I: Introduction to task scheduling

Scheduling

Example:
2 processors

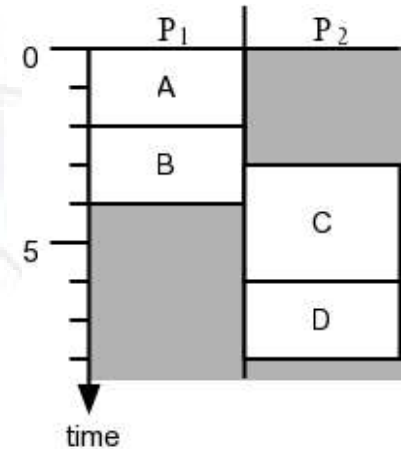


I: Introduction to task scheduling

Scheduling constraints

Schedule definitions: DAG: $G(V, E)$, node n , edge e

- start time: $t_s(n)$; finish time: $t_f(n)$
- processor assignment: $proc(n)$



Constraints:

- Processor constraint:
 $proc(n_i) = proc(n_j) \Rightarrow t_s(n_i) \geq t_f(n_j) \text{ or } t_s(n_j) \geq t_f(n_i)$
- Precedence constraint:
for all edges e_{ji} of E (from n_j to n_i)
 $t_s(n_i) \geq t_f(n_j) + c(e_{ji})$

I: Introduction to task scheduling

Static Task Scheduling

Temporal and spatial assignment of sub-tasks to processors at compile time

Goal: find schedule with shortest schedule length (makespan)
=> NP-hard problem

Scheduling heuristics

- List scheduling ↩
- Clustering
- Duplication scheduling
- Genetic algorithms

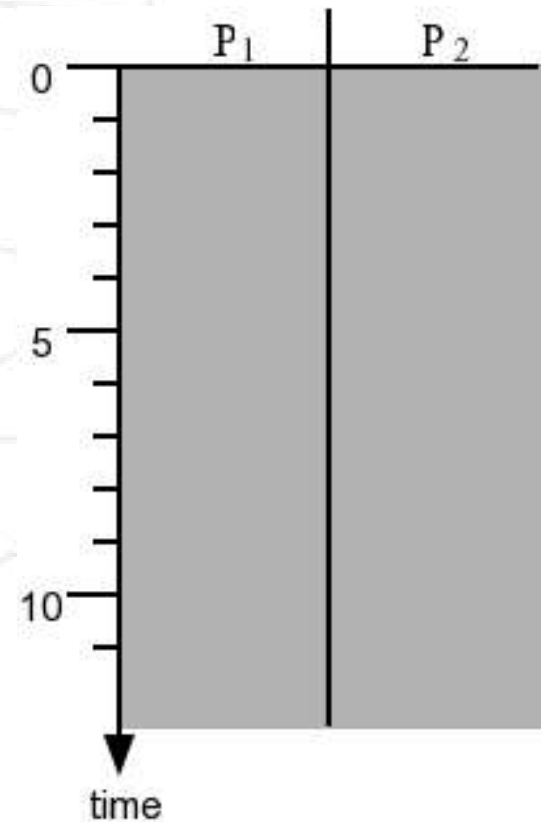
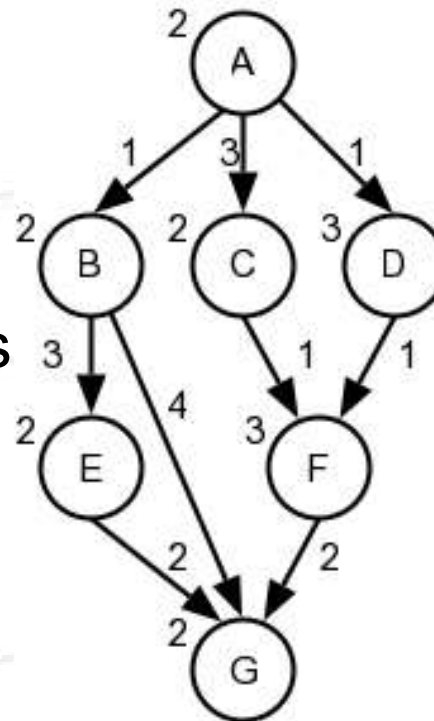
I: Introduction to task scheduling

List Scheduling

1. Order nodes of DAG according to a priority, while respecting their dependences
2. Iterate over node list from 1.) and schedule every node to the processor that allows its earliest start time.

Example:

Node order: A,C,D,F,B,E,G



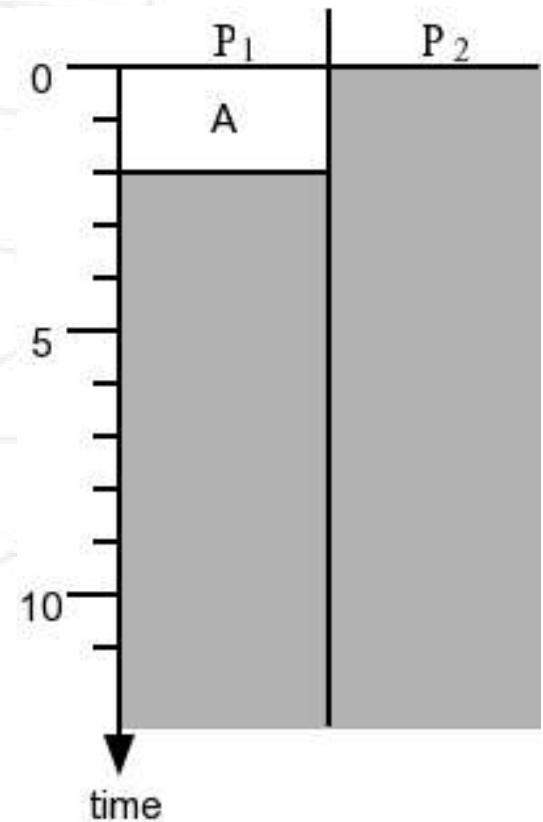
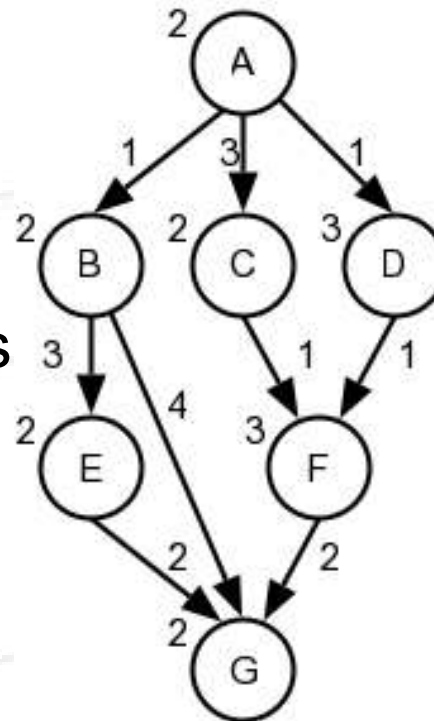
I: Introduction to task scheduling

List Scheduling

1. Order nodes of DAG according to a priority, while respecting their dependences
2. Iterate over node list from 1.) and schedule every node to the processor that allows its earliest start time.

Example:

Node order: A,C,D,F,B,E,G



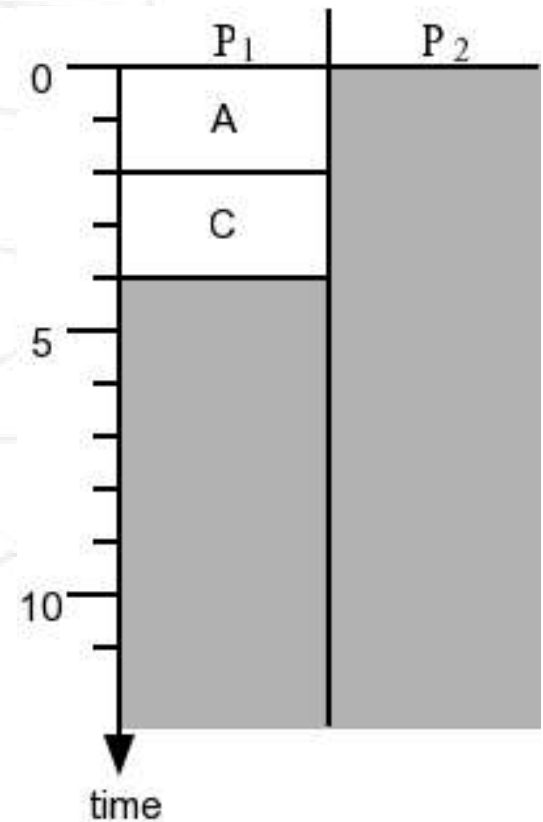
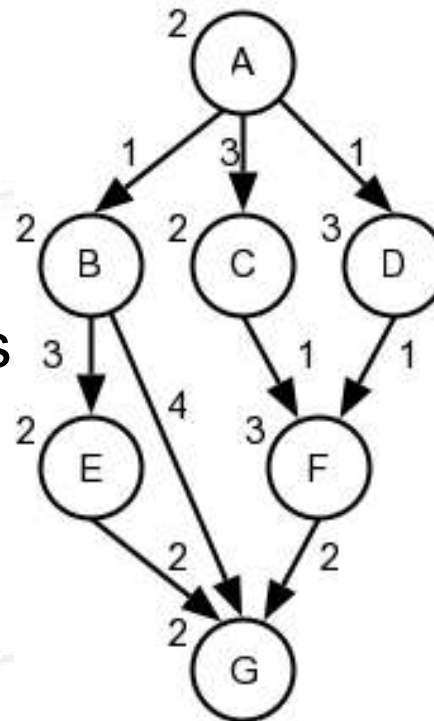
I: Introduction to task scheduling

List Scheduling

1. Order nodes of DAG according to a priority, while respecting their dependences
2. Iterate over node list from 1.) and schedule every node to the processor that allows its earliest start time.

Example:

Node order: A,C,D,F,B,E,G



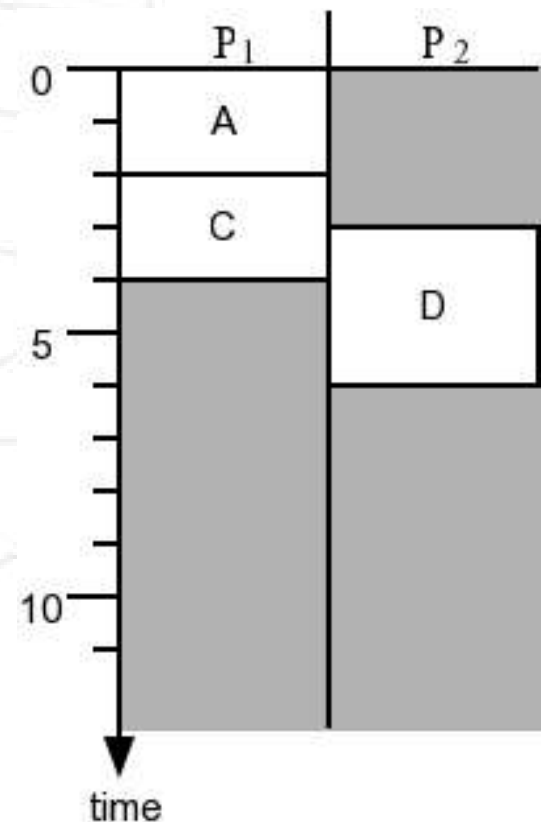
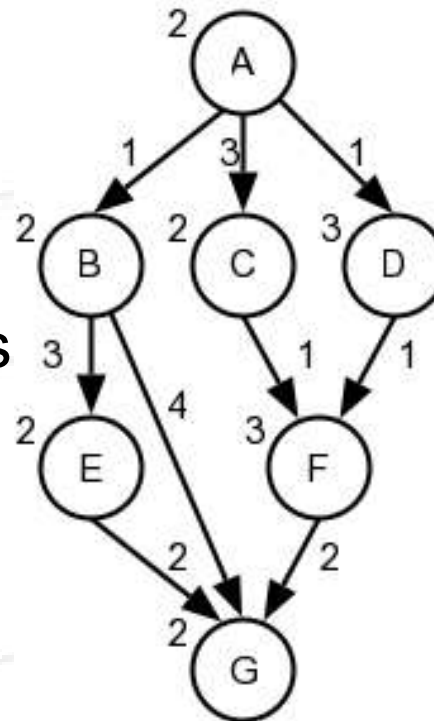
I: Introduction to task scheduling

List Scheduling

1. Order nodes of DAG according to a priority, while respecting their dependences
2. Iterate over node list from 1.) and schedule every node to the processor that allows its earliest start time.

Example:

Node order: A,C,D,F,B,E,G



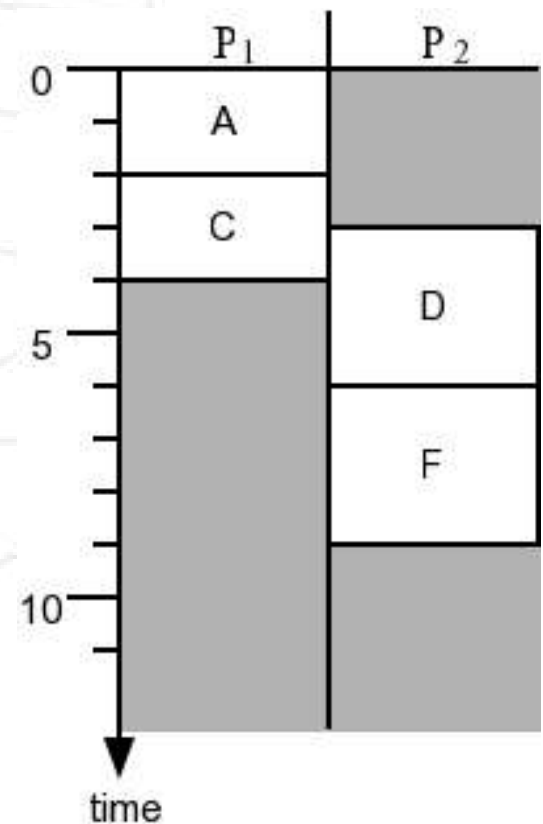
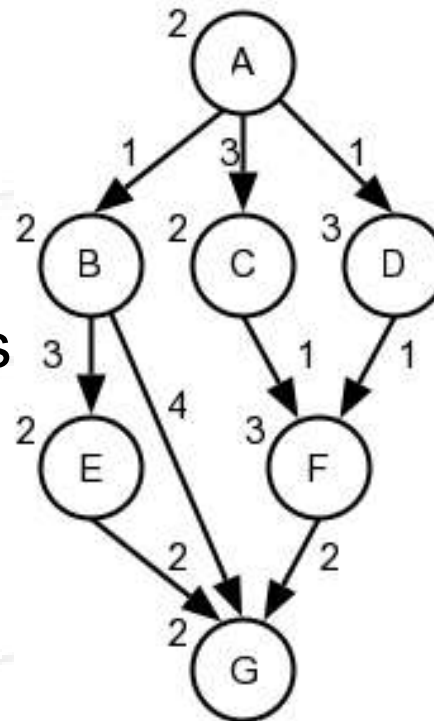
I: Introduction to task scheduling

List Scheduling

1. Order nodes of DAG according to a priority, while respecting their dependences
2. Iterate over node list from 1.) and schedule every node to the processor that allows its earliest start time.

Example:

Node order: A,C,D,F,B,E,G



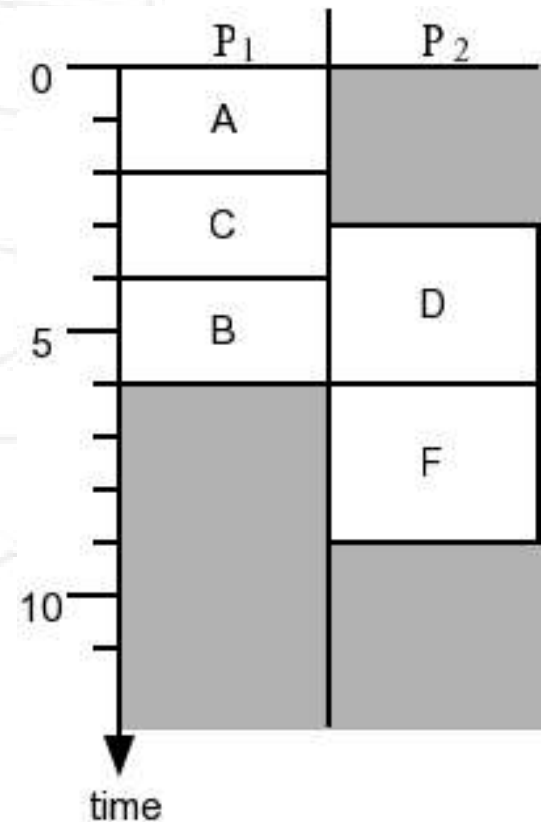
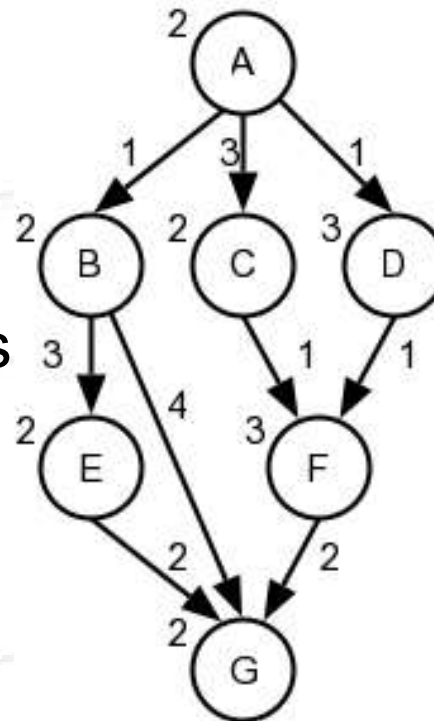
I: Introduction to task scheduling

List Scheduling

1. Order nodes of DAG according to a priority, while respecting their dependences
2. Iterate over node list from 1.) and schedule every node to the processor that allows its earliest start time.

Example:

Node order: A,C,D,F,B,E,G



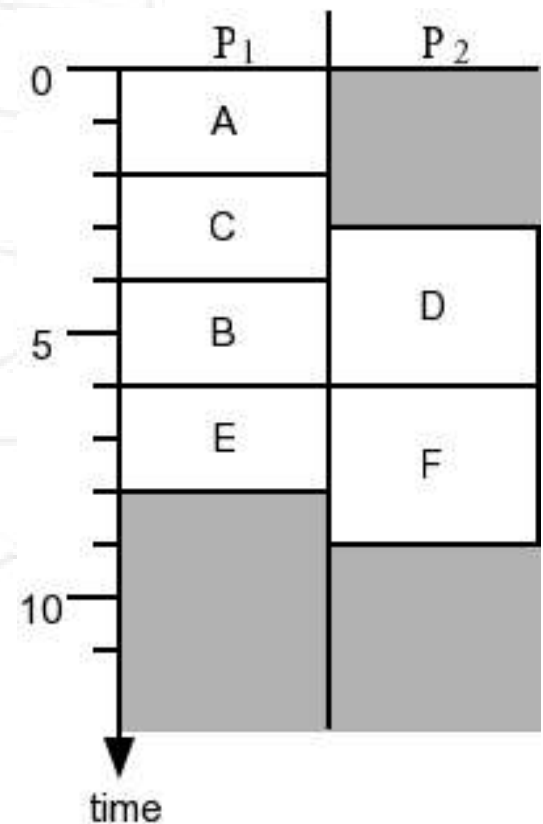
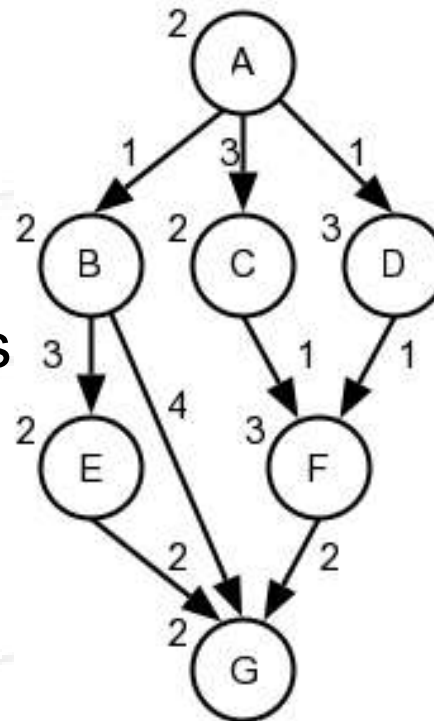
I: Introduction to task scheduling

List Scheduling

1. Order nodes of DAG according to a priority, while respecting their dependences
2. Iterate over node list from 1.) and schedule every node to the processor that allows its earliest start time.

Example:

Node order: A,C,D,F,B,E,G



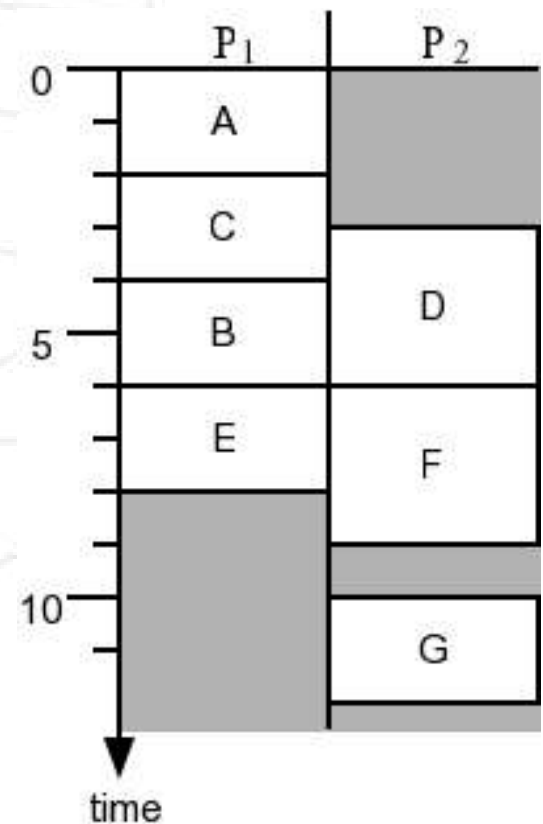
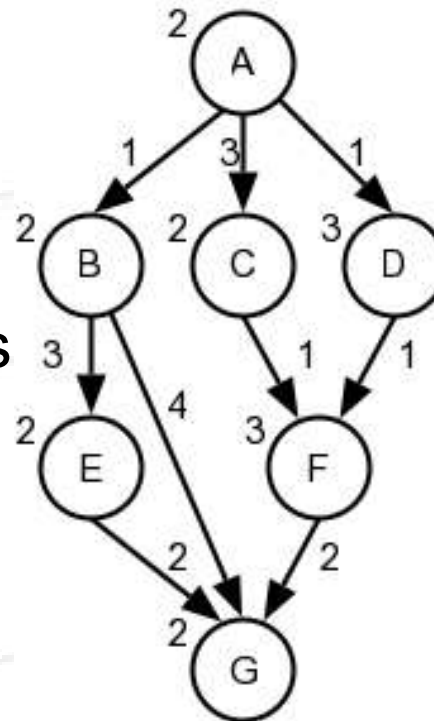
I: Introduction to task scheduling

List Scheduling

1. Order nodes of DAG according to a priority, while respecting their dependences
2. Iterate over node list from 1.) and schedule every node to the processor that allows its earliest start time.

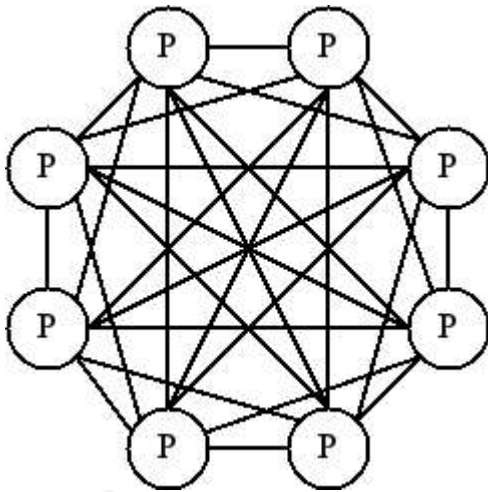
Example:

Node order: A,C,D,F,B,E,G



Classic system model of task scheduling

system model



e.g. 8 processors

Properties:

- Dedicated system
- Dedicated processors
- Zero-cost local communication
- Communication subsystem
- Concurrent communication ↩
- Fully connected ↩

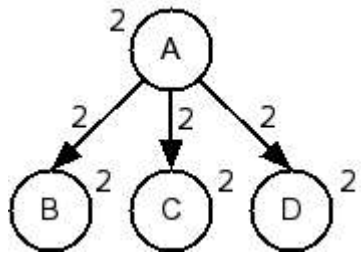


II: Contention scheduling

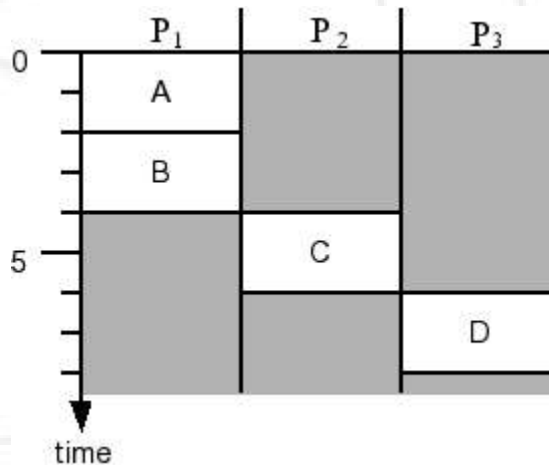
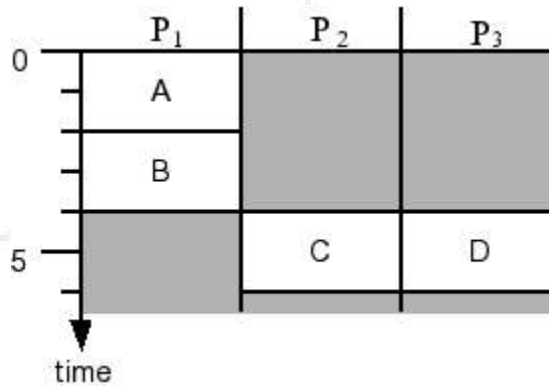
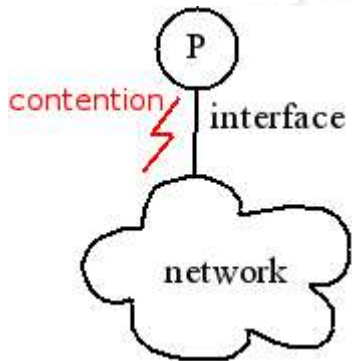
II: Contention scheduling

Communication contention

contention example



classic
model



- End-point contention
 - For Interface

- Most networks *not* fully connected



- Network contention
 - For network links

II: Contention scheduling

Network model

Sophisticated network graph:

Vertices: processors (P) and switches (S)

- Static and dynamic networks
- End-point and network contention

Edges: communication links (L)

- Undirected edges

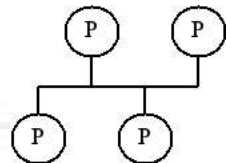
– Half duplex 

- Directed edges

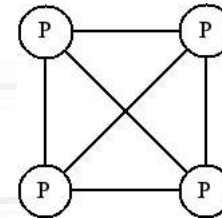
– Full duplex 

- Hyperedges

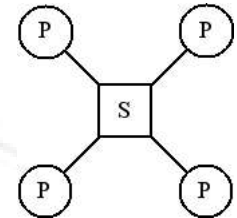
– Bus



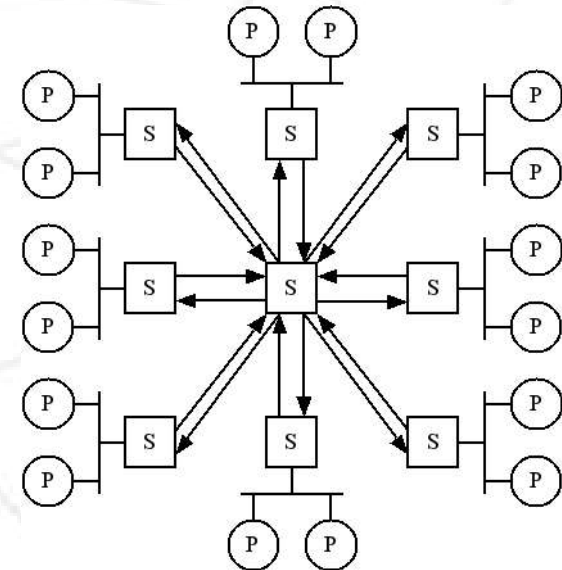
fully connected



switched LAN



example: 8 dual-processor cluster

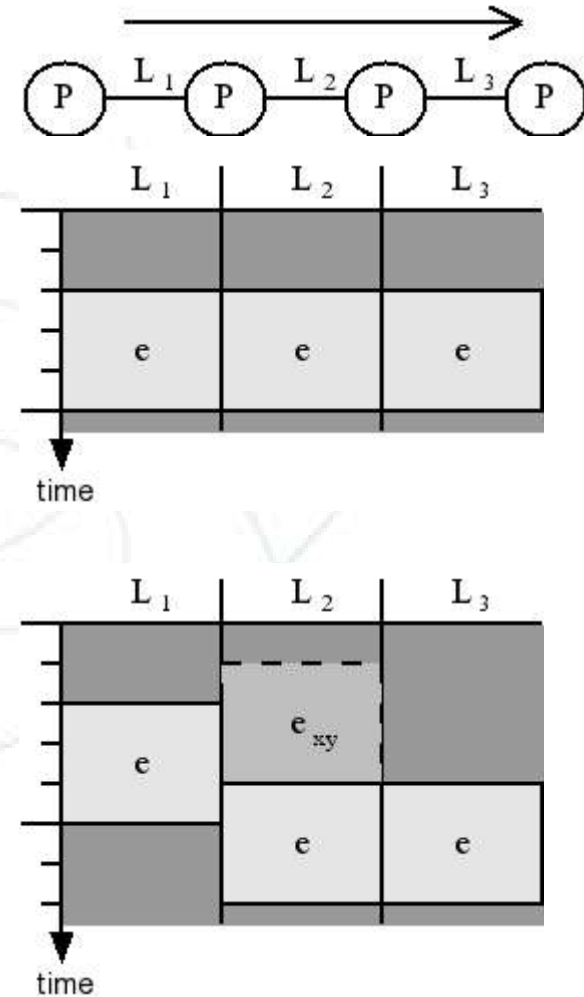


II: Contention scheduling

Edge scheduling

Scheduling of edges on links (L)

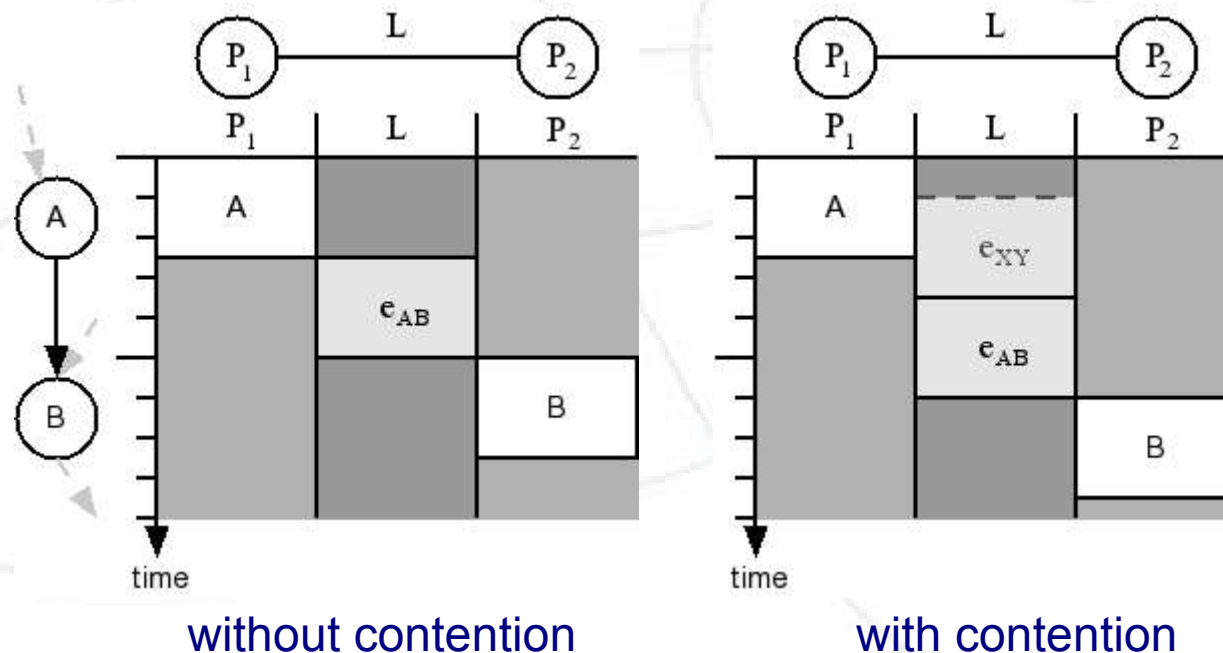
- Likewise nodes on processors
- Routing:
 - Policies
 - System dependent routing algorithm returns route, i.e. $\langle L_1, L_2, L_3 \rangle$
- Edge scheduled on each link of route
 - Independent of edge types
- Causality
- Heterogeneity



II: Contention scheduling

Contention aware scheduling

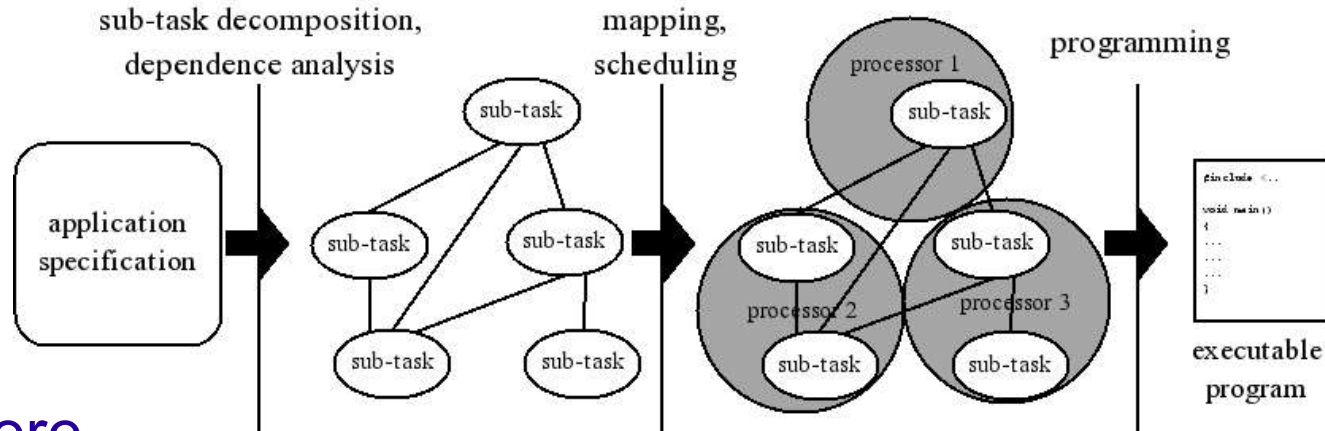
- Target system represented as network graph
- Integration of edge scheduling into task scheduling
 - Only impact on start time of node:
 - $t_s(n_i) \geq t_f(e_{ji})$ (precedence constraint)



III: Generating the task graph

III: Generating the Task Graph

Sub-task decomposition and dependence analysis



Until here

- Task Graph is considered as given

How to generate Task Graph for an application specification/program?

- Dependence analysis of program (\Rightarrow compiler)
 - Very difficult in its general form
- Annotating a program

III: Generating the Task Graph

Using OpenMP like directives

OpenMP

- Open standard for shared-memory programming
- Compiler directives used with FORTRAN, C/C++, Java
- Thread based

Examples (in C)

```
#pragma omp parallel for
for (i=0; i<=n+1; i++) {
...
}
```

```
#pragma omp parallel sections
{
#pragma omp section {
...
}
#pragma omp section {
...
}
...
}
```

III: Generating the Task Graph

Tasks/Task directives

Introduction of new directives: `tasks/task`

- Like sections with finer granularity
- Dependences and computation weights can be specified

```
#pragma omp parallel tasks
{
  #pragma omp task A 1 {
    ...
  }
  #pragma omp task B 2 dependsOn(A) {
    ...
  }
  ...
}
```

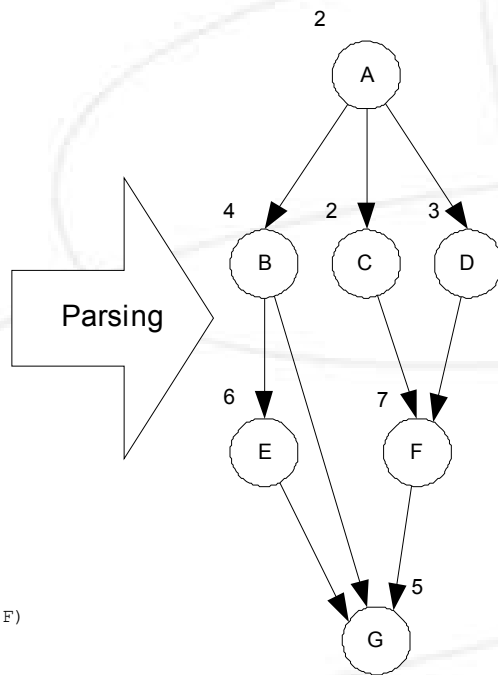
`Tasks/task` are transformed into `sections/section` with the aid of task scheduling

III: Generating the Task Graph JompX

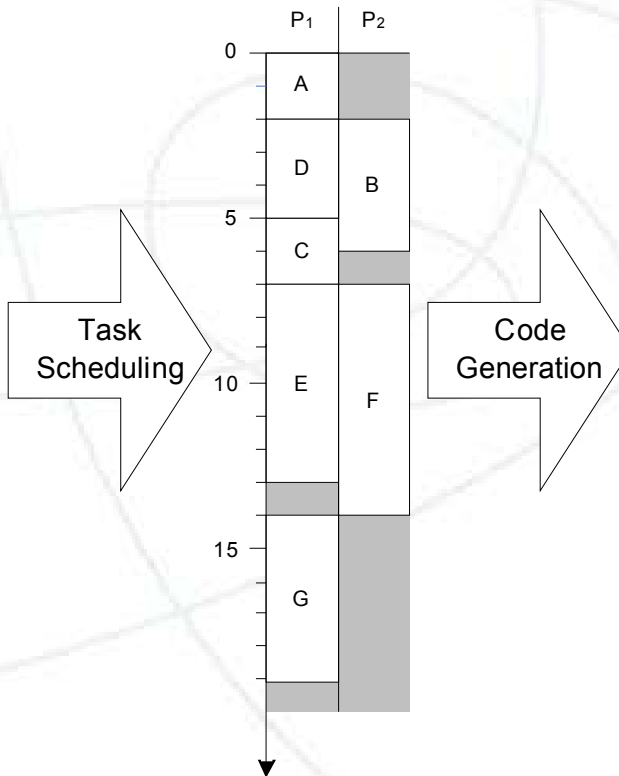
Source-To-Source compiler

- Java/OpenMP+task directives => Java/OpenMP

```
//omp parallel tasks
{
    //omp task A 2
    {
        Block_Code_A
    }
    //omp task B 4 dependsOn (A)
    {
        Block_Code_B
    }
    //omp task C 2 dependsOn (A)
    {
        Block_Code_C
    }
    //omp task D 3 dependsOn (A)
    {
        Block_Code_D
    }
    //omp task E 6 dependsOn (B)
    {
        Block_Code_E
    }
    //omp task F 7 dependsOn (C,D)
    {
        Block_Code_F
    }
    //omp task G 5 dependsOn (B,E,F)
    {
        Block_Code_G
    }
}
```



Tasks Graph representation



Schedule of the tasks graph

```
boolean taskADone = false;
boolean taskDDone = false;
boolean taskCDone = false;
boolean taskBDone = false;
boolean taskFDone = false;
//omp parallel sections
{
    //omp section
    {
        Block_Code_A
        taskADone = true;
        Block_Code_D
        taskDDone = true;
        Block_Code_C
        taskCDone = true;
        while (!taskBDone){}
        Block_Code_E
        while (!taskBDone){}
        while (!taskFDone){}
        Block_Code_G
    }
    //omp section
    {
        while (!taskADone){}
        Block_Code_B
        taskBDone = true;
        while (!taskCDone){}
        while (!taskDDone){}
        Block_Code_F
        taskFDone = true;
    }
}
```

Codes with sections directives

Code with tasks directives

III: Generating the Task Graph

Task Graph visualisation in Eclipse IDE

The screenshot displays the Eclipse IDE interface with the following components:

- Left Panel (Editor):** Contains the Java code for `sample.java`. The code includes OpenMP annotations for parallel tasks and dependencies.
- Center Panel:** Displays a task graph visualization. Nodes are green circles labeled with IDs and weights (e.g., ID=A, W=2). Edges represent dependencies between tasks.
- Right Panel (Outline):** Contains a table listing object IDs and their weights.
- Bottom Panel:** Shows the Problems view with a list of errors.

Java Code (Left):

```
8 {
9   int testInt;
10  //omp parallel tasks
11  {
12    boolean test = false;
13    //omp task A 2
14    {
15      //Block_Code_A
16      for(int i = 0; i < 45; i++){
17        System.out.println();
18      }
19    }
20    //omp task B 4 dependsOn(A)
21    {
22      if(test)
23        System.out.println("still");
24    }
25    //omp task C 2 dependsOn(A)
26    {
27      testInt = 110;
28    }
29    //omp task D 3 dependsOn(A)
30    {
31      while(testInt > 0){
32        System.out.println("testin");
33        testInt--;
34      }
35    }
36    //omp task E 6 dependsOn(B)
37    {
38      switch(testInt){
39        case 1:
40          return;
41        case 0:
42          System.out.println("0");
43      }
44    }
45  }
46 }
```

Task Graph (Center):

```
graph TD
  A["ID=A  
W=2"] --> B["ID=B  
W=4"]
  A --> C["ID=C  
W=2"]
  A --> D["ID=D  
W=3"]
  A --> E["ID=E  
W=6"]
  A --> F["ID=F  
W=7"]
  A --> G["ID=G  
W=5"]
  B --> D
  B --> E
  C --> D
  C --> E
  D --> F
  D --> G
  E --> F
  E --> G
  F --> G
```

Object ID and Weight Table (Right):

Object ID	Weight
B	4
C	2
D	3
E	6
F	7
G	5
edge1	-1
edge2	-1
edge3	-1
edge10	-1
edge11	-1
edge12	-1
edge4	-1
edge7	-1
edge13	-1
edge5	-1
edge14	-1
edge6	-1
edge15	-1
edge8	-1
edge9	-1

Problems View (Bottom):

- Iterable - java.lang
- Iterator - java.util
- Iterator - javax.swing.text.html.HTMLDocument
- Iterator - sun.text.CompactShortArray
- IteratorPool - com.sun.org.apache.xpath.internal.axes

Left:
Annotated Java Code

Right:
Visualisation of
dependence structure

Conclusion

My research in Parallel Computing

Task Scheduling

O. Sinnen, “Task Scheduling for Parallel Systems”, John Wiley, 2007

Reconfigurable hardware

Desktop parallelisation => Nasser Giacaman



Contact

Department of Electrical and Computer Engineering

University of Auckland

www.ece.auckland.ac.nz/~sinnen/

o.sinnen@auckland.ac.nz