Task Scheduling for Parallel Systems

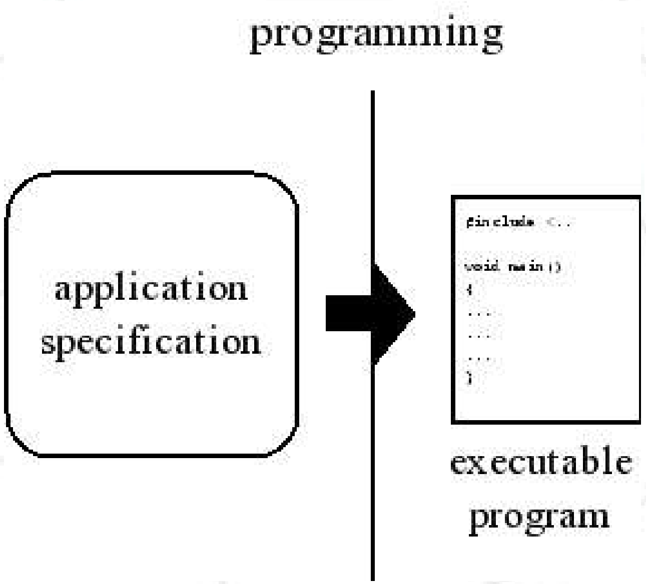
Oliver Sinnen

Electrical and Computer Engineering
University of Auckland

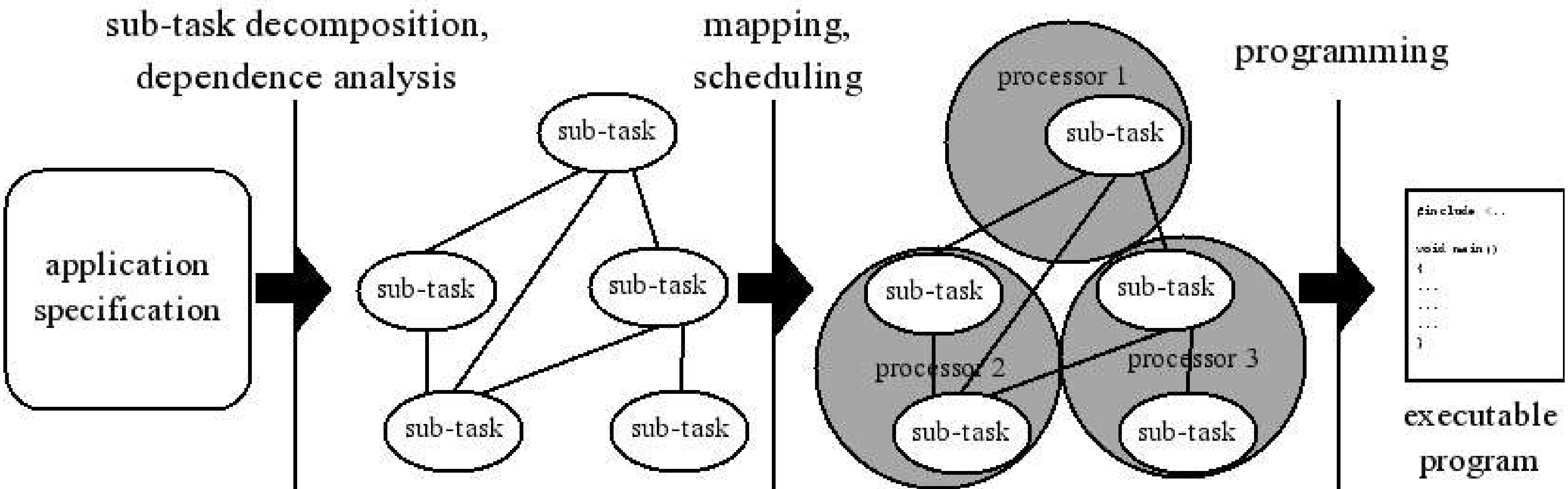
www.ece.auckland.ac.nz/~sinnen/
o.sinnen@auckland.ac.nz

Parallel Programming

Sequential programming



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

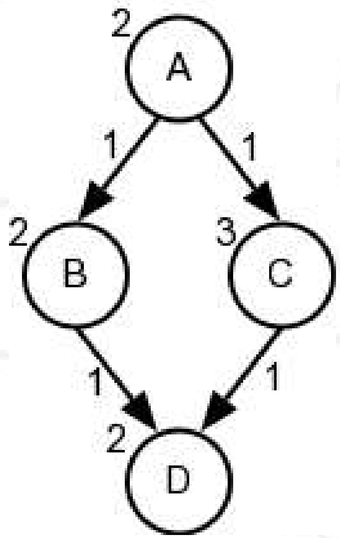
- III: Generating the Task Graph
  - Extending OpenMP

Graph representation of program

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

Example:

task graph (DAG)



- Graph representation of program
- Input of task scheduling

directed acyclic graph (DAG)

node (n): sub-task

edge (e): dependence (communication)

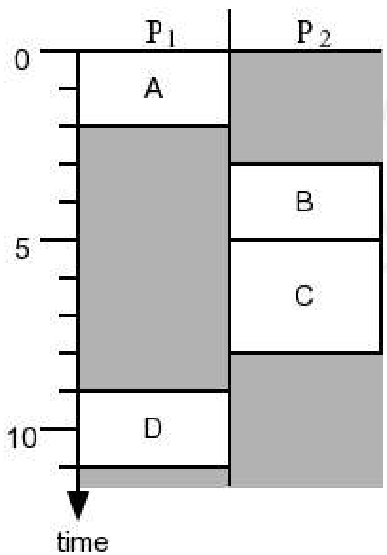
weight: computation *w(n)* or communication time *c(e)*

Scheduling

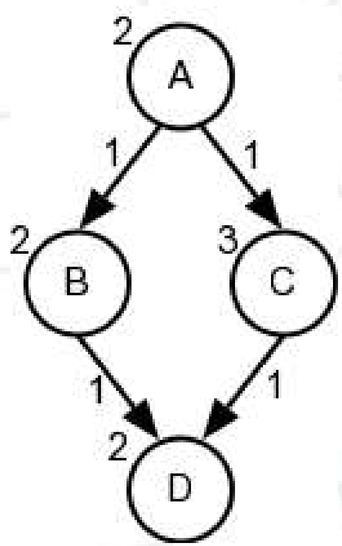
Example:

2 processors

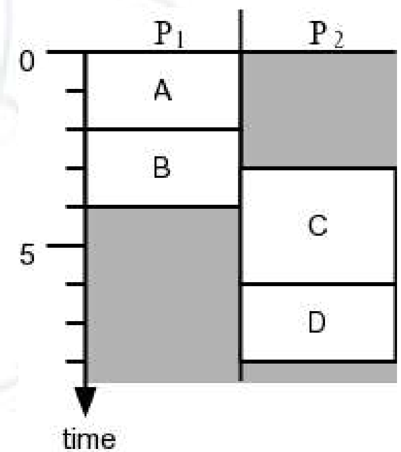
+



ex.



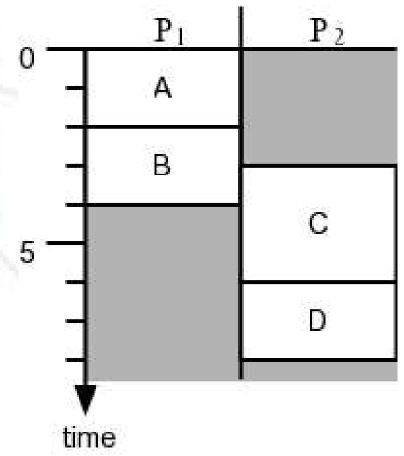
ex.



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:

  $proc(n_i)=proc(n_j) \;\; \Rightarrow \;\; t_s(n_i) \geq t_f(n_j)$ 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

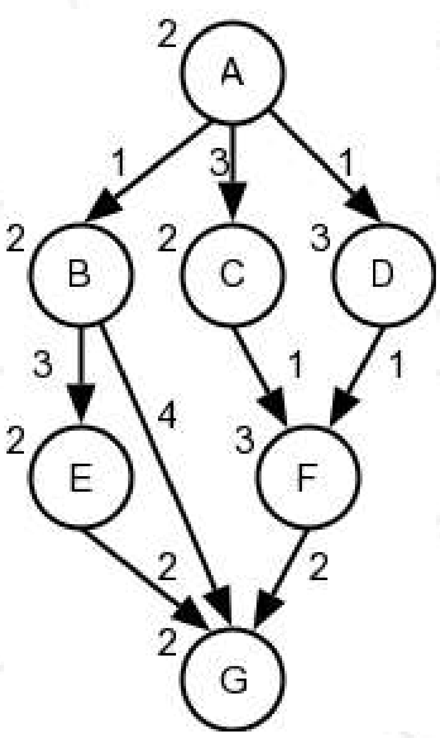
- List scheduling
- Clustering
- Duplication scheduling
- Genetic algorithms

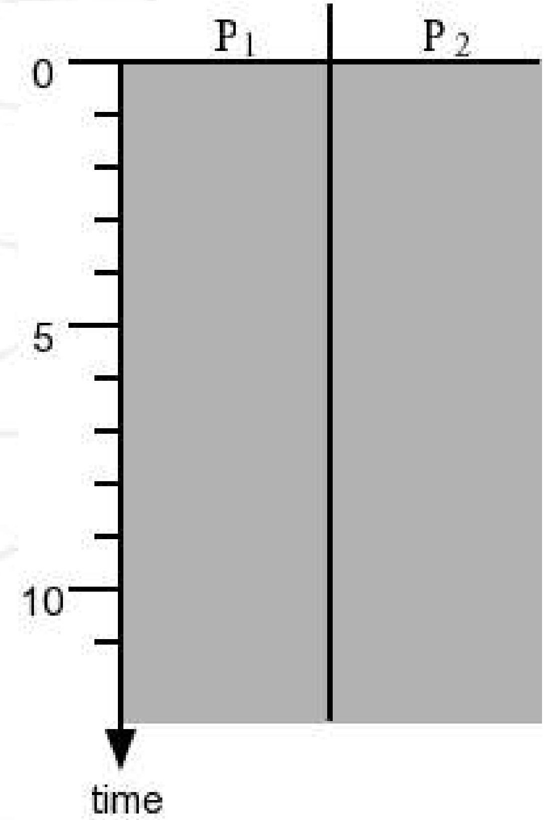
List Scheduling

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



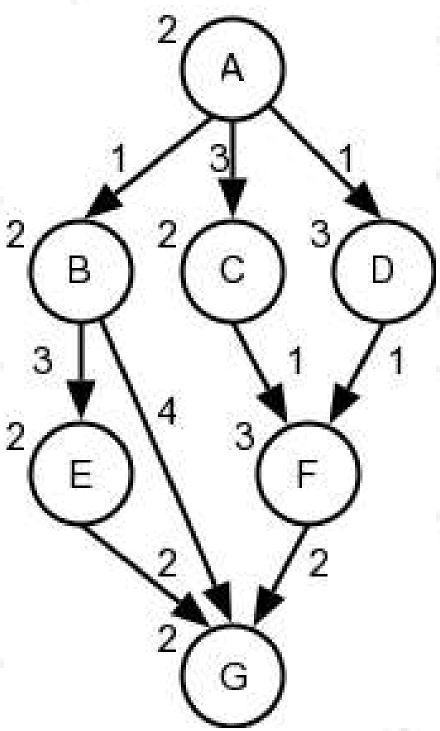


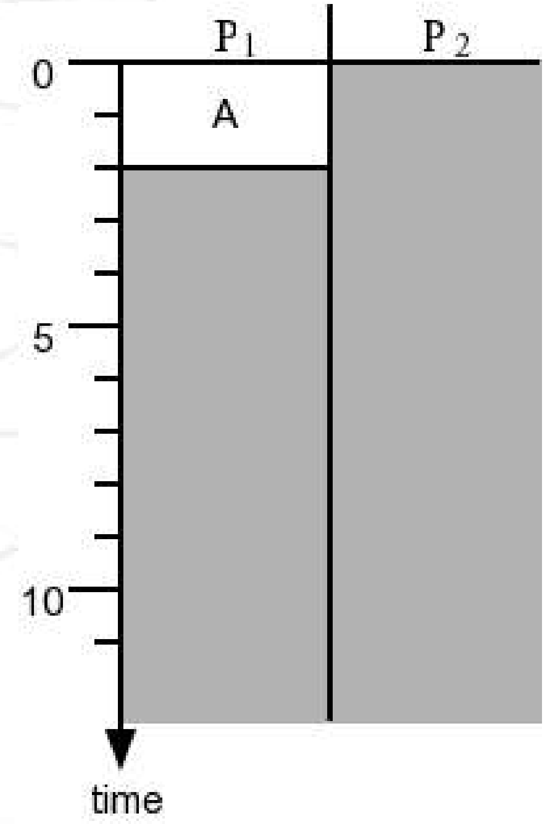
List Scheduling

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



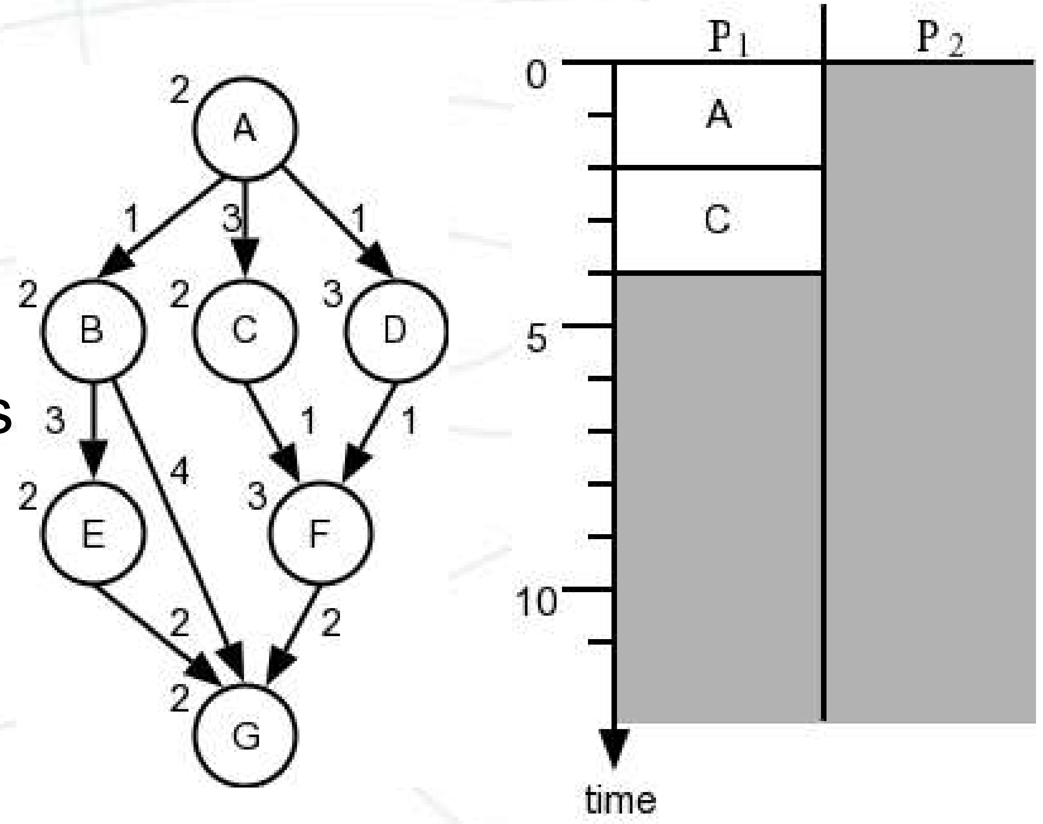


List Scheduling

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

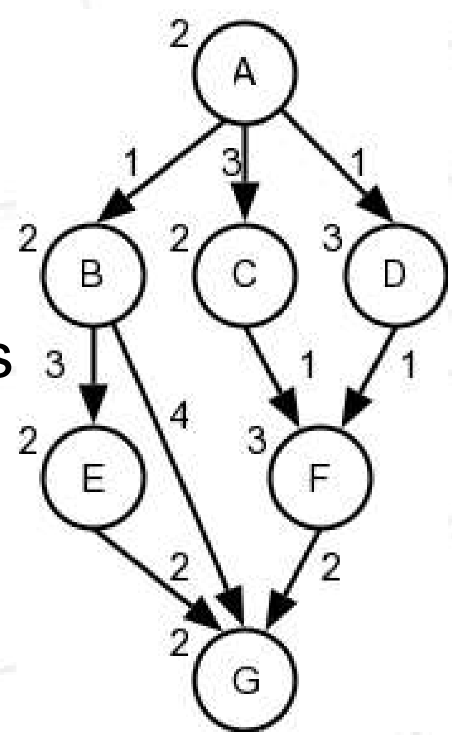


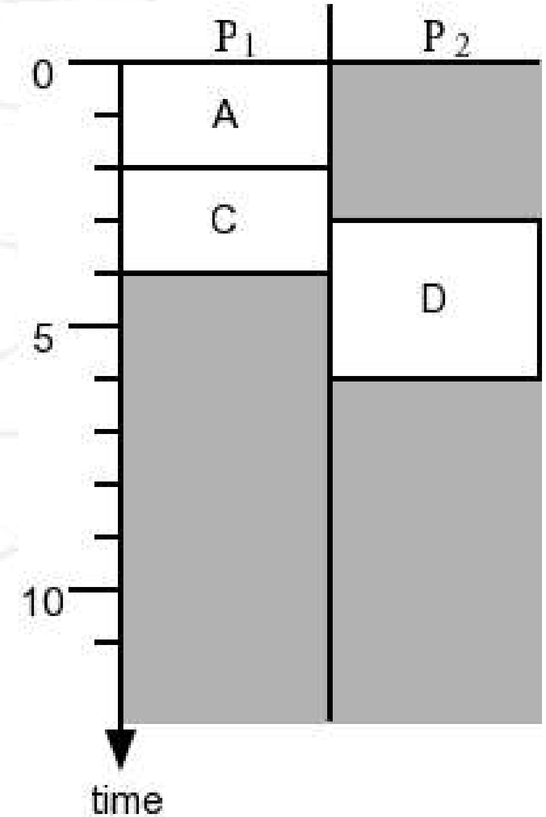
List Scheduling

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



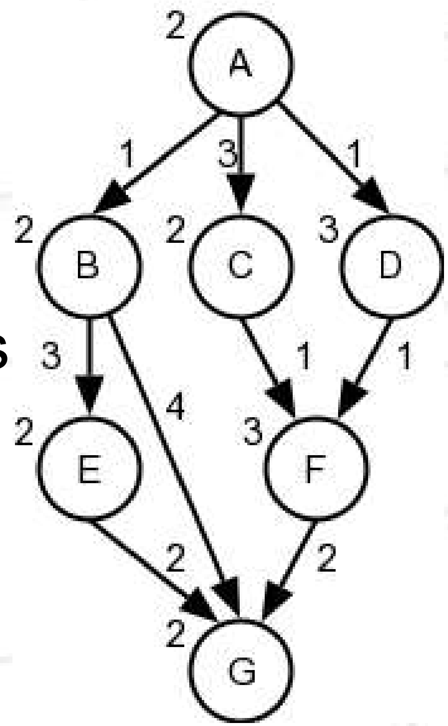


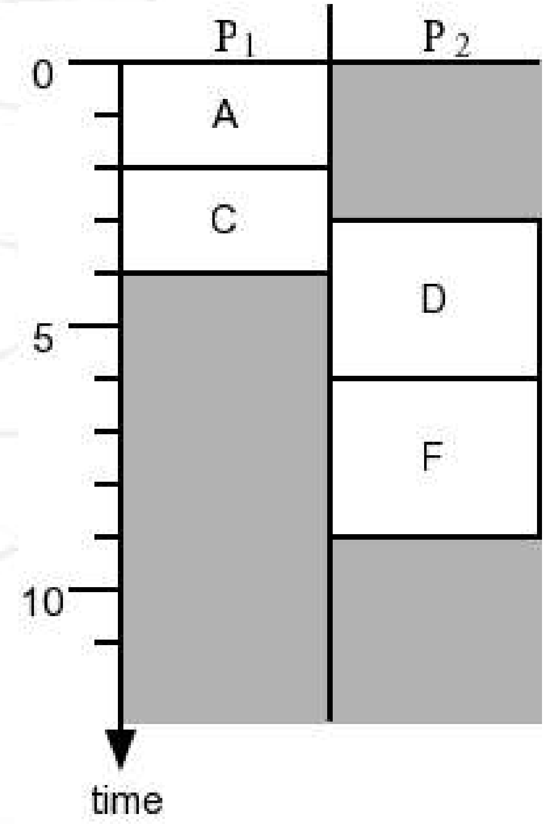
List Scheduling

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



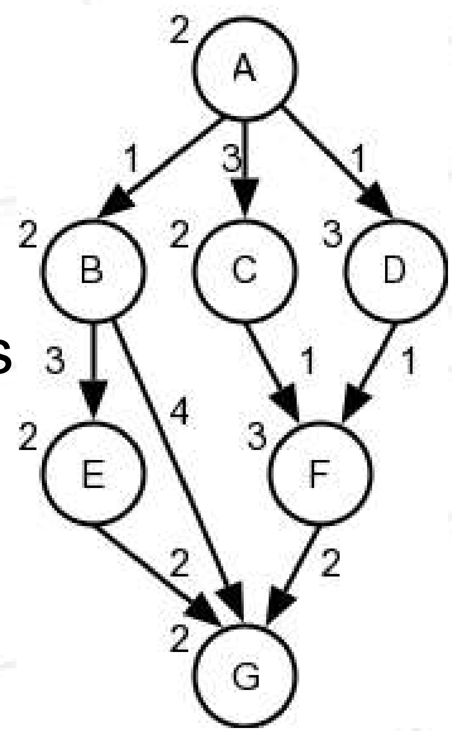


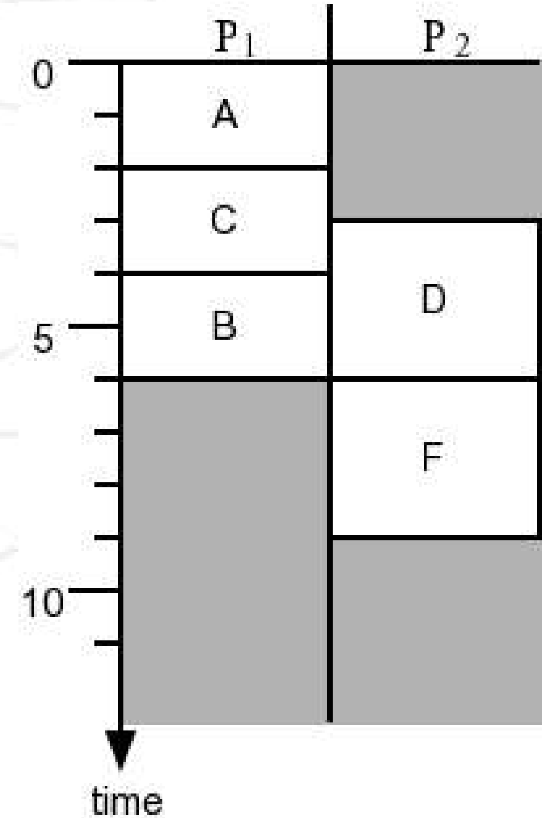
List Scheduling

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



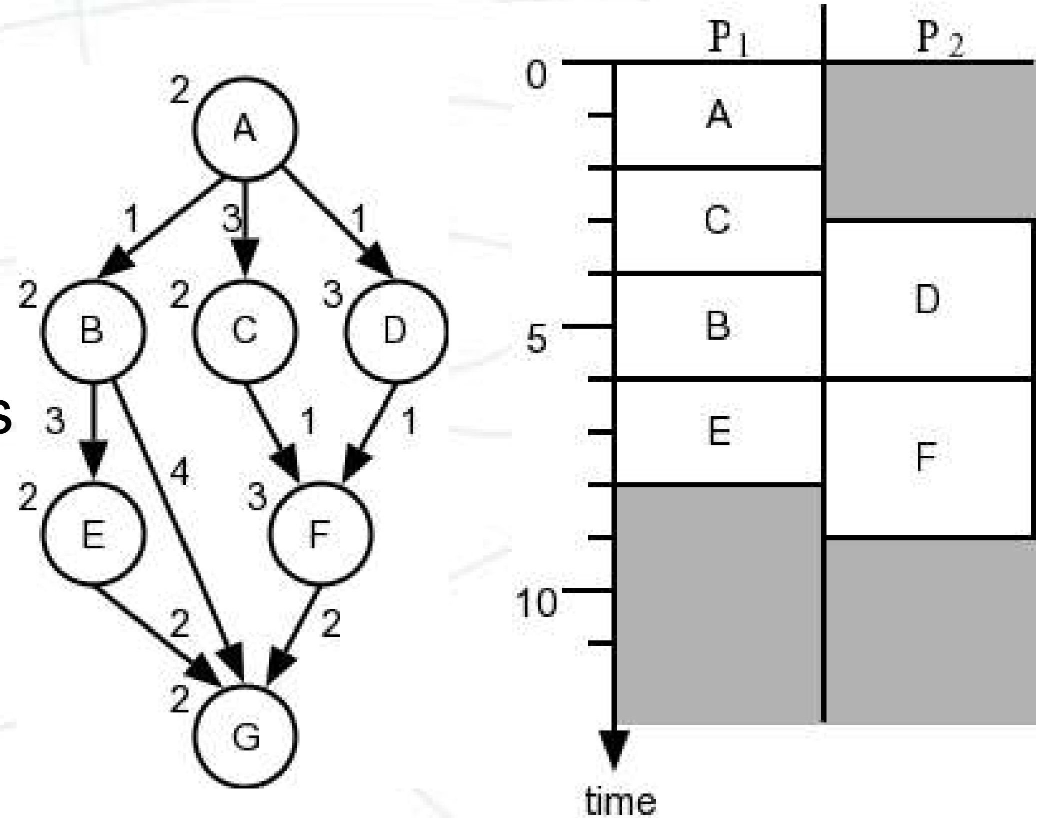


List Scheduling

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

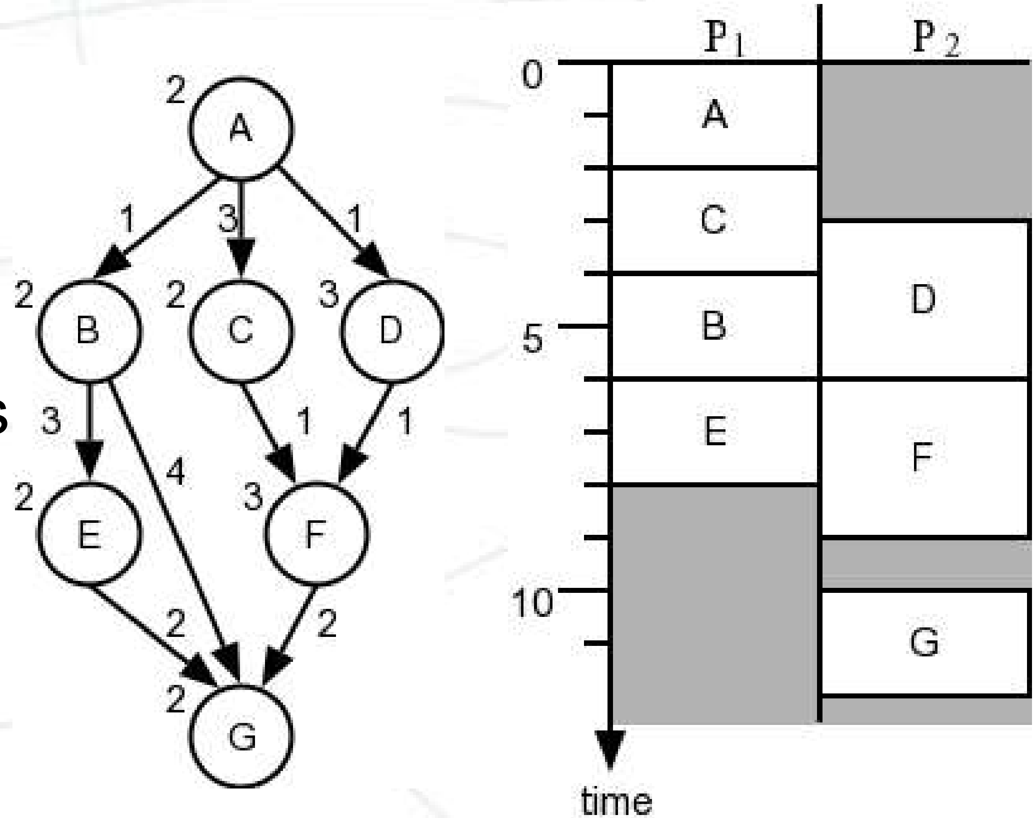


List Scheduling

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.

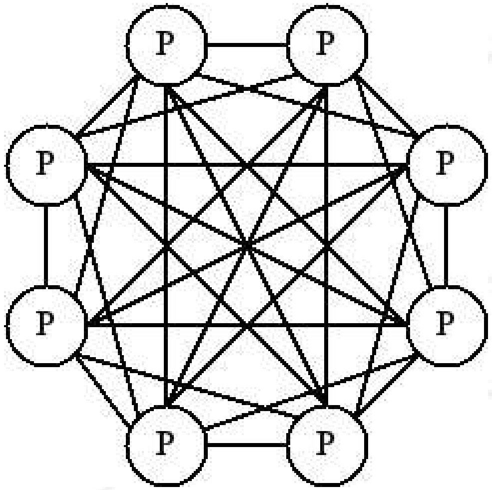
Example:

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



Classic system model of task scheduling

system model



e.g. 8 processors

Properties:

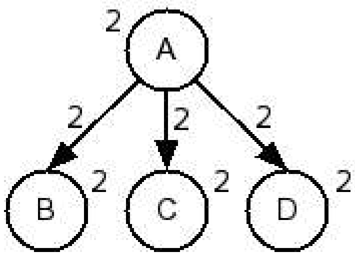
- Dedicated system
- Dedicated processors
- Zero-cost local communication
- Communication subsystem
- Concurrent communication
- Fully connected

II: Contention scheduling

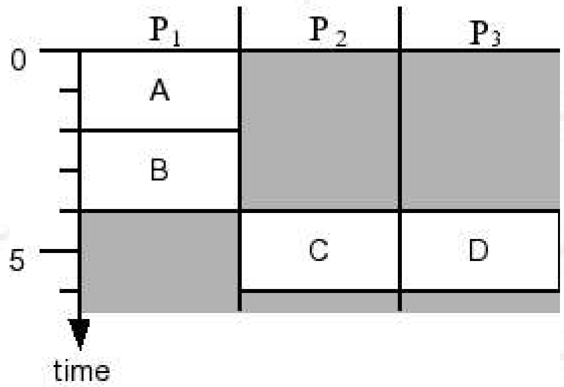
II: Contention scheduling

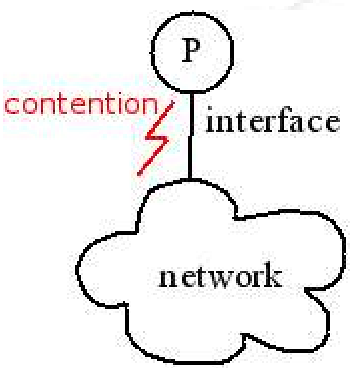
Communication contention

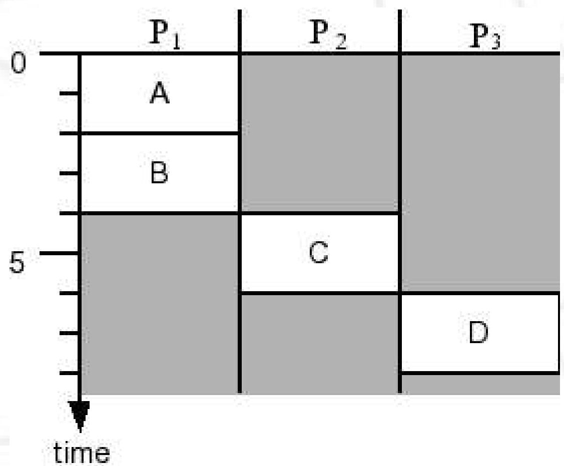
contention example



classic model







- End-point contention
  - For Interface
- Most networks *not* fully connected
- Network contention
  - For network links

II: Contention scheduling

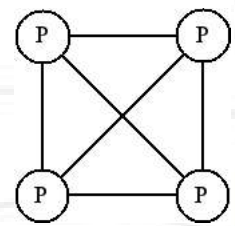
Network model

Sophisticated network graph:

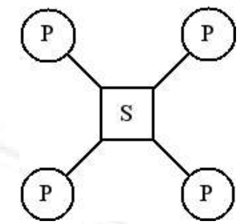
Vertices: processors (P) and switches (S)

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

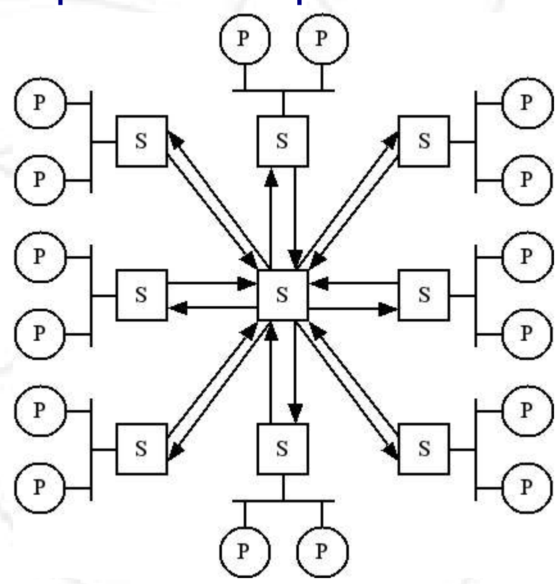
fully connected



switched LAN



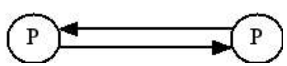
example: 8 dual-processor cluster



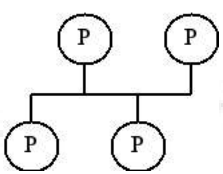
Edges: communication links (L)

- Undirected edges
  - Half duplex

- Directed edges
  - Full duplex



- Hyperedges
  - Bus

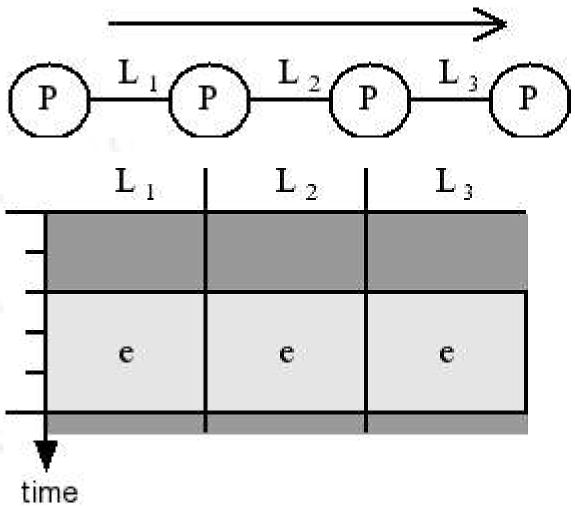


II: Contention scheduling

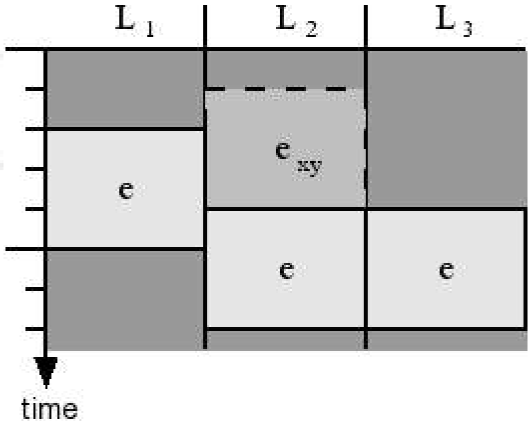
Edge scheduling

Scheduling of edges on links (L)

- Likewise nodes on processors

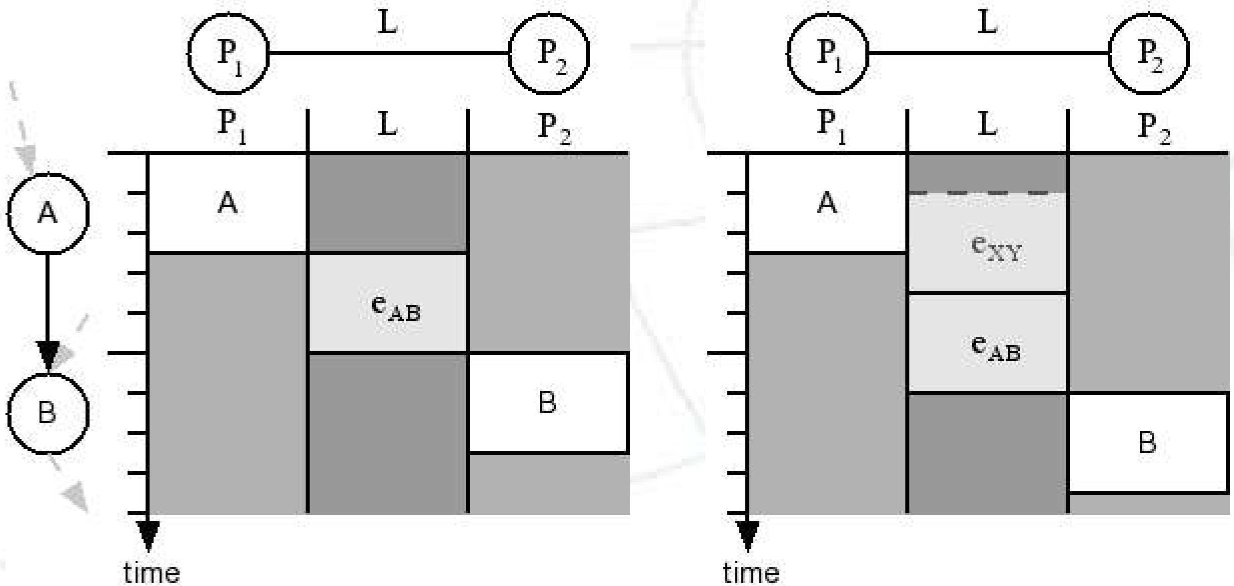


- Routing:
  - Policies
  - System dependent routing algorithm returns route, i.e. <L_1, L_2, L_3>
- Edge scheduled on each link of route
  - Independent of edge types
- Causality
- Heterogeneity



Contention aware scheduling

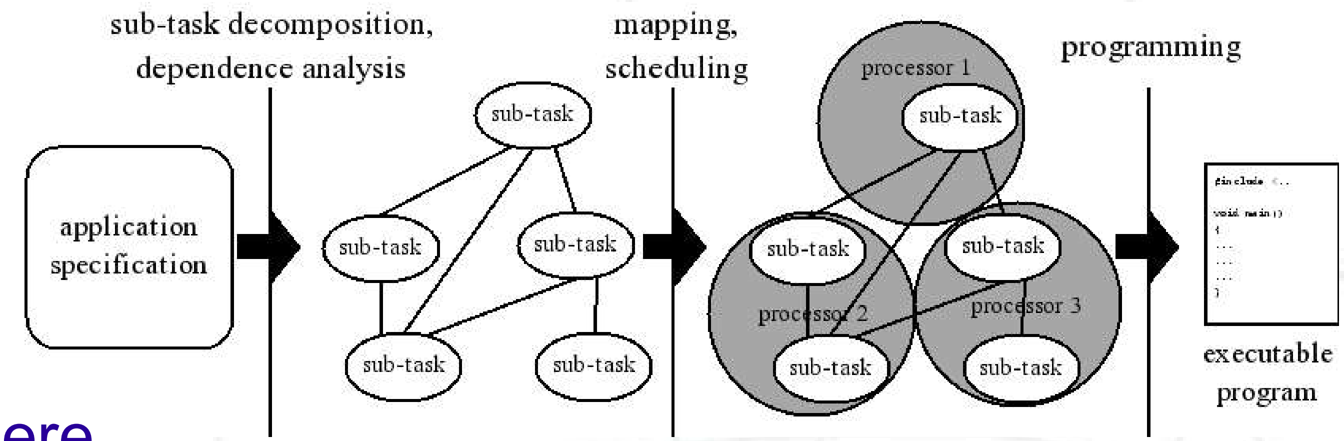
- Target system represented as network graph
- Integration of edge scheduling into task scheduling
  - Only impact on start time of node:
  - $t_s(n_i) \geq t_f(e_{ji})$ (precedence constraint)



without contention with contention

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 (=> 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

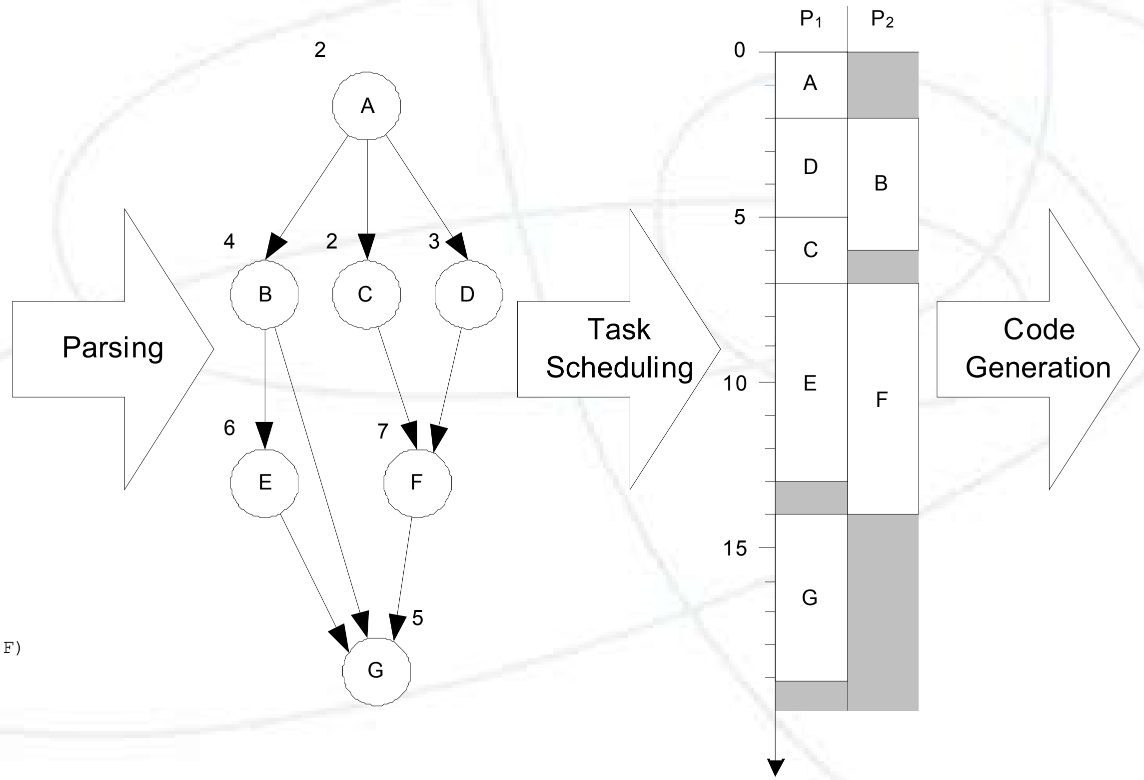
JompX

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
    }
}
```

Code with tasks directives



Tasks Graph representation

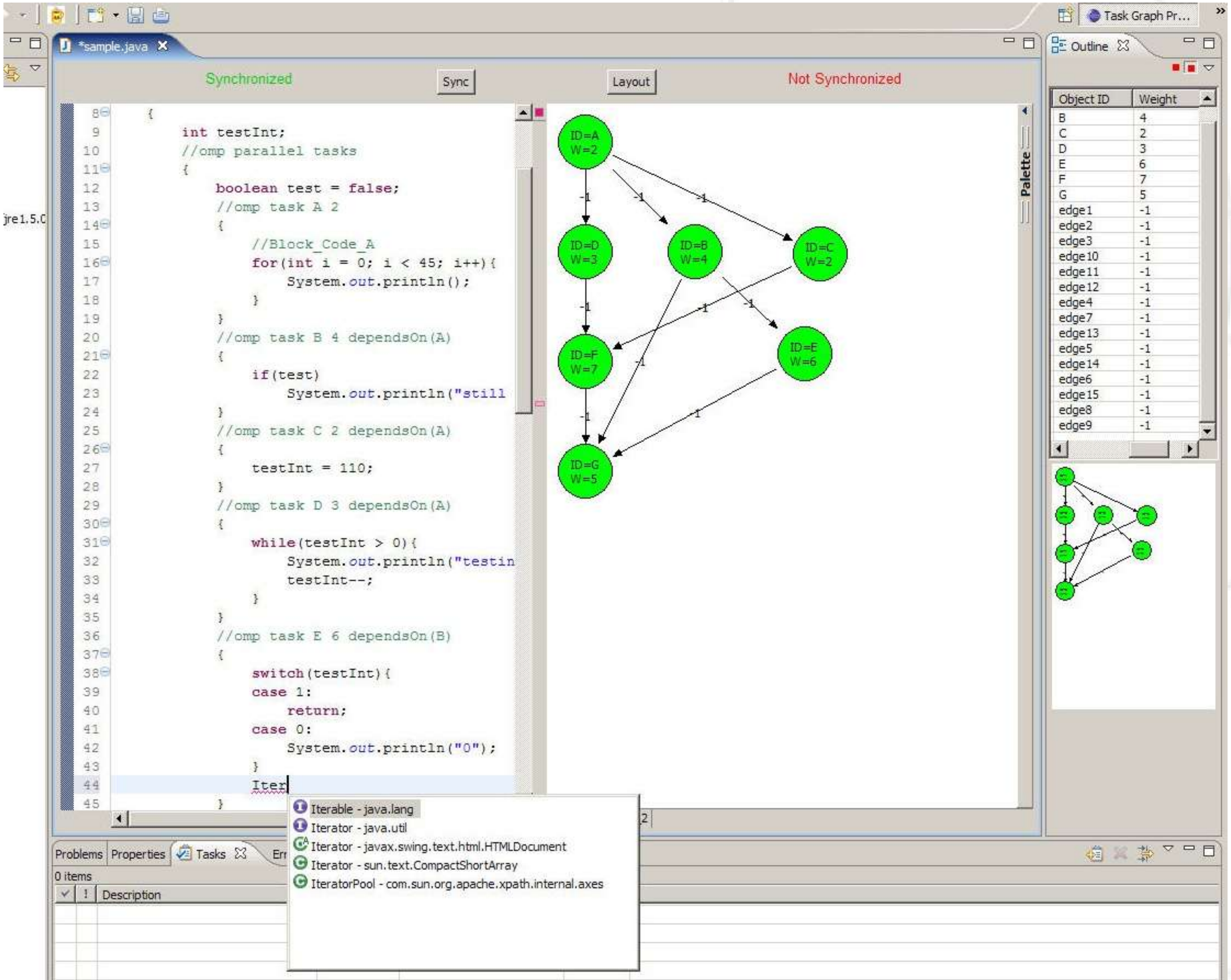
Schedule of the tasks graph

```
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

III: Generating the Task Graph

Task Graph visualisation in Eclipse IDE



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