Algorithms and scheduling techniques for clusters and grids

Yves Robert

École Normale Supérieure de Lyon
Yves.Robert@ens-lyon.fr
http://graal.ens-lyon.fr/~yrobert

joint work with
Olivier Beaumont, Anne Benoit, Larry Carter, Henri Casanova,
Jack Dongarra, Jeanne Ferrante, Arnaud Legrand, Loris Marchal, Frédéric Vivien

Murcia, May 26, 2008
Evolution of parallel machines

From (good old) parallel architectures . . .
Evolution of parallel machines

... to heterogeneous clusters ...
Evolution of parallel machines

... and to large-scale grid platforms?
Evolution of parallel machines

...and to large-scale grid platforms?

Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines
Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines

On heterogeneous platforms, it gets worse
New platforms, new problems, new solutions

**Target platforms:** Large-scale heterogeneous platforms 
(networks of workstations, clusters, collections of clusters, grids, ...)

**New problems**

- Heterogeneity of processors (CPU power, memory)
- Heterogeneity of communication links
- Irregularity of interconnection networks
- Non-dedicated platforms

Need to adapt algorithms and scheduling strategies: new objective functions, new models
New platforms, new problems, new solutions

Target platforms: Large-scale heterogeneous platforms
(networks of workstations, clusters, collections of clusters, grids, ...)

New problems

- Heterogeneity of processors (CPU power, memory)
- Heterogeneity of communication links
- Irregularity of interconnection networks
- Non-dedicated platforms

Need to adapt algorithms and scheduling strategies: new objective functions, new models

Yves.Robert@ens-lyon.fr February 8, 2008
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion

Yves.Robert@ens-lyon.fr February 8, 2008
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion
Independent chunks

- $B$ independent equal-size tasks
- $p$ processors $P_1, P_2, \ldots, P_p$
- $w_i = \text{time for } P_i \text{ to process a task}$

**Intuition:** load of $P_i$ proportional to its speed $1/w_i$

Assign $n_i$ tasks to $P_i$

**Objective:** minimize $T_{exe} = \max \sum_{i=1}^{p} n_i = B (n_i \times w_i)$
Dynamic programming

With 3 processors: $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$
Dynamic programming

With 3 processors: $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$

<table>
<thead>
<tr>
<th>Task</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$T_{exe}$</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

**With 3 processors:** \( w_1 = 3, \ w_2 = 5, \text{ and } w_3 = 8 \)

---

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( T_{\text{exe}} )</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

---

Yves.Robert@ens-lyon.fr  February 8, 2008
Dynamic programming

With 3 processors: $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$
Dynamic programming

With 3 processors: $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$

<table>
<thead>
<tr>
<th>Task</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$T_{exe}$</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>
Dynamic programming

**With 3 processors:** \( w_1 = 3, \ w_2 = 5, \text{ and } w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( T_{\text{exe}} )</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

**With 3 processors:** \( w_1 = 3, w_2 = 5, \text{ and } w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>T_{exe}</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

With 3 processors: \( w_1 = 3, \ w_2 = 5, \ \text{and} \ w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>T_{exe}</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
## Dynamic programming

**With 3 processors:** \( w_1 = 3, w_2 = 5, \text{ and } w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( T_{\text{exe}} )</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

With 3 processors: $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$

<table>
<thead>
<tr>
<th>Task</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$T_{\text{exe}}$</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming

With 3 processors: \( w_1 = 3, \ w_2 = 5, \text{ and } w_3 = 8 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( T_{\text{exe}} )</th>
<th>Selected proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Static versus dynamic

- Greedy (demand-driven) would have done a perfect job
- Would even be better (possible variations in processor speeds)

Static assignment required useless thinking 😞
Static versus dynamic

- Greedy (demand-driven) would have done a perfect job
- Would even be better (possible variations in processor speeds)

Static assignment required *useless thinking* 😞
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
     - Matrix product (ScaLAPACK)
     - Matrix product (master-slave)
     - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion
Coping with dependences

A simple finite difference problem

- Iteration space: 2D rectangle of size $N_1 \times N_2$
- Dependences between tiles $\{ (1, 0), (0, 1) \}$
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution...
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution
Use column-wise allocation to enhance locality

Stepwise execution
 Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution
Use column-wise allocation to enhance locality

Stepwise execution
Allocation strategy (2/3)

- With column-wise allocation,

\[ T_{opt} \approx \frac{N_1 \times N_2}{\sum_{i=1}^{\pi} \frac{1}{w_i}}. \]

- Greedy (demand-driven) allocation \( \Rightarrow \text{slowdown} \) ?!
- Execution progresses at the pace of the slowest processor 😞
Allocation strategy (2/3)

- With column-wise allocation,
  \[ T_{\text{opt}} \approx \frac{N_1 \times N_2}{\sum_{i=1}^{p} \frac{1}{w_i}}. \]

- Greedy (demand-driven) allocation ⇒ slowdown ?!
- Execution progresses at the pace of the slowest processor 😞
With 3 processors, \( w_1 = 3 \), \( w_2 = 5 \), and \( w_3 = 8 \):

\[
T_{\text{exe}} \approx \frac{8}{3} N_1 N_2 \approx 2.67 N_1 N_2
\]

\[
T_{\text{opt}} \approx \frac{120}{79} N_1 N_2 \approx 1.52 N_1 N_2
\]
Periodic static allocation (1/2)

With 3 processors, \( w_1 = 3 \), \( w_2 = 5 \), and \( w_3 = 8 \):

Assigning blocks of \( B = 10 \) columns, \( T_{\text{exe}} \approx 1.6 \ N_1 N_2 \)
Periodic static allocation (2/2)

- $L = \text{lcm}(w_1, w_2, \ldots, w_p)$
  
  **Example:** $L = \text{lcm}(3, 5, 8) = 120$

- $P_1$ receives first $n_1 = L/w_1$ columns, $P_2$ next $n_2 = L/w_2$ columns, and so on

- Period: block of $B = n_1 + n_2 + \ldots + n_p$ contiguous columns
  
  **Example:** $B = n_1 + n_2 + n_3 = 40 + 24 + 15 = 79$

- **Change schedule:**
  - Sort processors so that $n_1 w_1 \leq n_2 w_2 \leq \ldots \leq n_p w_p$
  - Process horizontally within blocks

- **Optimal 😊**
Lesson learnt?

With different-speed processors . . .
. . . we need to think (design static schedules)

. . . but implementation may remain dynamic 😊

Example: demand-driven allocation of blocks of adequate size

. . . well, in some cases it gets truly complicated 😞
Lesson learnt?

With different-speed processors …
… we need to think (design static schedules)

… but implementation may remain dynamic 😊

Example: demand-driven allocation of blocks of adequate size

… well, in some cases it gets truly complicated 😞
Lesson learnt?

With different-speed processors . . .
. . . we need to think (design static schedules)

. . . but implementation may remain dynamic 😊

Example: demand-driven allocation of blocks of adequate size

. . . well, in some cases it gets truly complicated 😞
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion
Why revisit matrix-product?

- A fundamental computational kernel (the mother of parallel algorithms)
- Archetype of a tightly-coupled application
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm
Use $q \times q$ blocks to harness efficiency of Level 3 BLAS
ScaLAPACK algorithm on (homogeneous) 2D grids (1/2)

- \( C = AB \) on a \( p_1 \times p_2 \) processor grid
- Granularity: one element = one square \( q \times q \) block
- Each matrix is partitioned into \( p_1 \times p_2 \) rectangles
- Each processor is responsible for updating its rectangle
- Outer product version: at each step,
  - a column of blocks is communicated (broadcast) horizontally
  - a row of blocks is communicated (broadcast) vertically
Matrix product on a $3 \times 4$ homogeneous 2D-grid
Matrix product on a $3 \times 4$ heterogeneous 2D-grid
2D load balancing (1/2)

**Objective:** \( \max \ r_i \times w_{ij} \times c_j \leq 1 \ \{(\sum_{i=1}^{p_1} r_i) \times (\sum_{j=1}^{p_2} c_j)\} \)

Maximize total number of elements processed within one time unit.
Given $p$ processors, how to arrange them along a 2D grid of size $p_1 \times p_2 \leq p$ ...

... so as to optimally load-balance the work of the processors

- Search among all possible arrangements of $p_1 \times p_2$ processors as a $p_1 \times p_2$ grid
- For each arrangement, solve optimization problem
- **NP-hard 😞**
Matrix product on heterogeneous clusters

Matrix product with 13 heterogeneous processors

Yves.Robert@ens-lyon.fr February 8, 2008
How to compute the *area* and *shape* of the $p$ rectangles?

- **Load-balancing computations** assign *areas* proportional to speeds
- **Minimizing communication overhead** choose *shapes*:
  - total communication volume
    \[
    \hat{C} = \sum_{i=1}^{p} (h_i + v_i)
    \]
    *sum* of the half perimeters of the $p$ rectangles
  - for parallel communications:
    \[
    \hat{M} = \max_{i=1}^{p} (h_i + v_i)
    \]

- Both problems NP-hard 😞
Optimization

How to compute the *area* and *shape* of the *p* rectangles?

- **Load-balancing computations** assign *areas* proportional to speeds

- **Minimizing communication overhead** choose *shapes*:
  - total communication volume

\[
\hat{C} = \sum_{i=1}^{p} (h_i + v_i)
\]

*sum* of the half perimeters of the *p* rectangles

- for parallel communications:

\[
\hat{M} = \max_{i=1}^{p} (h_i + v_i)
\]

- **Both problems NP-hard 😞**
Outline

1 Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2 Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3 Pipeline workflows

4 Models and real life

5 Conclusion
Why revisit matrix-product?

- A fundamental computational kernel (the mother of parallel algorithms)
- Archetype of a tightly-coupled application
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm
- Target platforms = heterogeneous clusters
- Target usage = speed up MATLAB-client
Why revisit matrix-product?

- A fundamental computational kernel (the mother of parallel algorithms)
- Archetype of a tightly-coupled application
- Well-understood for *homogeneous 2D-arrays of processors*
  - Cannon algorithm
  - ScaLAPACK outer product algorithm
- **Target platforms** = heterogeneous clusters
- **Target usage** = speed up MATLAB-client
Same application model

Use $q \times q$ blocks to harness efficiency of Level 3 BLAS
Platform model

- **Star network** master $M$ and $p$ workers $P_i$
- $X.w_i$ time-units for $P_i$ to execute a task of size $X$
- $X.c_i$ time-units for $M$ to send/rcv msg of size $X$ to/from $P_i$
- Master has no processing capability
- Enforce *one-port* model

Memory limitation: only $m_i$ buffers available for $P_i$;
→ at most $m_i$ blocks simultaneously stored on worker
Platform model

- **Star network** master $M$ and $p$ workers $P_i$
- $X.w_i$ time-units for $P_i$ to execute a task of size $X$
- $X.c_i$ time-units for $M$ to send/rcv msg of size $X$ to/from $P_i$
- Master has no processing capability
- Enforce *one-port* model

**Memory limitation:** only $m_i$ buffers available for $P_i$
$\rightarrow$ at most $m_i$ blocks simultaneously stored on worker
Strategy for allocating buffers

- **Natural memory management**
  - Assign one-third for each of $A$, $B$ and $C$
  - **Example:** $m = 21 \Rightarrow 7$ buffers per matrix

- **Optimal memory management**
  - Find largest $\mu$ s.t. $1 + \mu + \mu^2 \leq m$
  - Assign 1 buffer to $A$, $\mu$ to $B$ and $\mu^2$ to $C$
  - **Example:** $m = 21 \Rightarrow 1$ for $A$, 4 to $B$ and 16 to $C$
Strategy for allocating buffers

- Natural memory management
  - Assign one-third for each of $A$, $B$ and $C$
  - **Example:** $m = 21 \Rightarrow 7$ buffers per matrix

- Optimal memory management
  - Find largest $\mu$ s.t. $1 + \mu + \mu^2 \leq m$
  - Assign 1 buffer to $A$, $\mu$ to $B$ and $\mu^2$ to $C$
  - **Example:** $m = 21 \Rightarrow 1$ for $A$, 4 to $B$ and 16 to $C$
Example with $m = 21$
Example with $m = 21$
Example with $m = 21$
Example with $m = 21$
Example with $m = 21$

<table>
<thead>
<tr>
<th></th>
<th>$C_{11}$</th>
<th>$C_{12}$</th>
<th>$C_{13}$</th>
<th>$C_{14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{11}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B_{11}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_{21}$</td>
<td>$C_{22}$</td>
<td>$C_{23}$</td>
<td>$C_{24}$</td>
<td></td>
</tr>
<tr>
<td>$C_{31}$</td>
<td>$C_{32}$</td>
<td>$C_{33}$</td>
<td>$C_{34}$</td>
<td></td>
</tr>
<tr>
<td>$C_{41}$</td>
<td>$C_{42}$</td>
<td>$C_{43}$</td>
<td>$C_{44}$</td>
<td></td>
</tr>
</tbody>
</table>
Example with $m = 21$

<table>
<thead>
<tr>
<th></th>
<th>A$_{11}$</th>
<th>B$_{11}$</th>
<th>B$_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>C$_{11}$</td>
<td>C$_{12}$</td>
<td>C$_{13}$</td>
<td>C$_{14}$</td>
</tr>
<tr>
<td>C$_{21}$</td>
<td>C$_{22}$</td>
<td>C$_{23}$</td>
<td>C$_{24}$</td>
</tr>
<tr>
<td>C$_{31}$</td>
<td>C$_{32}$</td>
<td>C$_{33}$</td>
<td>C$_{34}$</td>
</tr>
<tr>
<td>C$_{41}$</td>
<td>C$_{42}$</td>
<td>C$_{43}$</td>
<td>C$_{44}$</td>
</tr>
</tbody>
</table>
Example with \( m = 21 \)

\[
\begin{array}{cccc}
A_{11} & C_{11} & C_{12} & C_{13} & C_{14} \\
C_{21} & C_{22} & C_{23} & C_{24} \\
C_{31} & C_{32} & C_{33} & C_{34} \\
C_{41} & C_{42} & C_{43} & C_{44} \\
B_{11} & B_{12} & B_{13} & B_{14} \\
\end{array}
\]
Example with $m = 21$

\[
\begin{array}{cccc}
A_{21} & B_{11} & B_{12} & B_{13} & B_{14} \\
C_{11} & C_{12} & C_{13} & C_{14} \\
C_{21} & C_{22} & C_{23} & C_{24} \\
C_{31} & C_{32} & C_{33} & C_{34} \\
C_{41} & C_{42} & C_{43} & C_{44}
\end{array}
\]
Example with $m = 21$

<table>
<thead>
<tr>
<th></th>
<th>$B_{11}$</th>
<th>$B_{12}$</th>
<th>$B_{13}$</th>
<th>$B_{14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{31}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$C_{11}$</td>
<td>$C_{12}$</td>
<td>$C_{13}$</td>
<td>$C_{14}$</td>
</tr>
<tr>
<td></td>
<td>$C_{21}$</td>
<td>$C_{22}$</td>
<td>$C_{23}$</td>
<td>$C_{24}$</td>
</tr>
<tr>
<td></td>
<td>$C_{31}$</td>
<td>$C_{32}$</td>
<td>$C_{33}$</td>
<td>$C_{34}$</td>
</tr>
<tr>
<td></td>
<td>$C_{41}$</td>
<td>$C_{42}$</td>
<td>$C_{43}$</td>
<td>$C_{44}$</td>
</tr>
</tbody>
</table>
Example with $m = 21$

<table>
<thead>
<tr>
<th></th>
<th>$B_{11}$</th>
<th>$B_{12}$</th>
<th>$B_{13}$</th>
<th>$B_{14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{41}$</td>
<td>$C_{11}$</td>
<td>$C_{12}$</td>
<td>$C_{13}$</td>
<td>$C_{14}$</td>
</tr>
<tr>
<td></td>
<td>$C_{21}$</td>
<td>$C_{22}$</td>
<td>$C_{23}$</td>
<td>$C_{24}$</td>
</tr>
<tr>
<td></td>
<td>$C_{31}$</td>
<td>$C_{32}$</td>
<td>$C_{33}$</td>
<td>$C_{34}$</td>
</tr>
<tr>
<td></td>
<td>$C_{41}$</td>
<td>$C_{42}$</td>
<td>$C_{43}$</td>
<td>$C_{44}$</td>
</tr>
</tbody>
</table>
Example with $m = 21$
Example with $m = 21$

```
A_{12} | x  | x  | x  | x  
C_{11} | C_{12} | C_{13} | C_{14} 
C_{21} | C_{22} | C_{23} | C_{24} 
C_{31} | C_{32} | C_{33} | C_{34} 
C_{41} | C_{42} | C_{43} | C_{44} 
```
Example with $m = 21$
Algorithm with identical workers

\[ c = 2, \ w = 4.5, \ \mu = 4, \ t = 100, \text{ enroll } \mathcal{P} = 5 \text{ workers} \]
Algorithm with identical workers

\[ c = 2, \ w = 4.5, \ \mu = 4, \ t = 100, \ \text{enroll} \ \mathcal{P} = 5 \ \text{workers} \]
Algorithm with identical workers

\[ c = 2, \; w = 4.5, \; \mu = 4, \; t = 100, \; \text{enroll} \; \mathfrak{P} = 5 \; \text{workers} \]
Algorithm with identical workers

\[ c = 2, \ w = 4.5, \ \mu = 4, \ t = 100, \ \text{enroll } \mathcal{P} = 5 \ \text{workers} \]
Algorithm with identical workers

$c = 2, \, w = 4.5, \, \mu = 4, \, t = 100, \text{ enroll } \mathfrak{P} = 5 \text{ workers}$
Algorithm with identical workers

\[ c = 2, \, \psi = 4.5, \, \mu = 4, \, t = 100, \text{ enroll } \psi = 5 \text{ workers} \]
Performance

- Communication-to-computation ratio:
  \[ \frac{2}{t} + \frac{2}{\mu} \rightarrow \frac{2}{\sqrt{m}} \]
  
- Close to lower bound
  
- Enroll \( \mathcal{P} \leq p \) workers, where

  \[ \mathcal{P} = \left\lfloor \frac{\mu w}{2c} \right\rfloor \]

  In the example, \( \mathcal{P} = \lceil 4.5 \rceil \)

- Typically, \( c = q^2 \tau_c \) and \( w = q^3 \tau_a \)
  
  → resource selection \( \mathcal{P} = \left\lfloor \frac{\mu q \tau_a}{2\tau_c} \right\rfloor \)
Algorithms for heterogeneous platforms

- Different memory patterns for workers

- Complicated resource selection
- Complicated communication ordering
- Complicated schedule
- ... but it works fine 😊 (see experiments in papers)
Different memory patterns for workers

Complicated resource selection

Complicated communication ordering

Complicated schedule

... but it works fine 😊 (see experiments in papers)
Lesson learnt?

Can provide efficient algorithms for tightly coupled applications but requires lots of efforts

...implementation cannot be demand-driven unless ready to pay huge performance degradation

Example: resource selection plus static ordering mandatory for heterogeneous platforms
Lesson learnt?

Can provide efficient algorithms for tightly coupled applications but requires lots of efforts

... implementation cannot be demand-driven unless ready to pay huge performance degradation

Example: resource selection plus static ordering mandatory for heterogeneous platforms
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion
Iterative algorithms

Initial data (typically, a matrix)

**Algorithm**

1. Each processor performs a computation on its data chunk
2. Each processor exchanges the “border” of its data chunk of data with its neighbors
3. Go back to Step 1

Questions

- Which processors should be used?
- What amount of data should they receive?
- How do we partition initial data set?
Iterative algorithms

Initial data (typically, a matrix)

**Algorithm**

1. Each processor performs a computation on its data chunk
2. Each processor exchanges the “border” of its data chunk of data with its neighbors
3. Go back to Step 1

**Questions**

- Which processors should be used?
- What amount of data should they receive?
- How do we partition initial data set?

Impact of network models
Slicing data

- Data: a 2-D array

- Uni-dimensional partitioning into vertical slices

- Consequences:
  - Borders and neighbors easily defined
  - Constant volume of data exchanged between neighbors: $D_c$
  - Processors virtually organized into a ring
Slicing data

- Data: a 2-D array

- Uni-dimensional partitioning into vertical slices

- Consequences:
  - Borders and neighbors easily defined
  - Constant volume of data exchanged between neighbors: $D_c$
  - Processors virtually organized into a ring
Slicing data

Data: a 2-D array

Uni-dimensional partitioning into vertical slices

Consequences:
1. Borders and neighbors easily defined
2. Constant volume of data exchanged between neighbors: $D_c$
3. Processors virtually organized into a ring
Slicing data

- Data: a 2-D array

![Diagram of slicing data]

- Uni-dimensional partitioning into vertical slices
- Consequences:
  1. Borders and neighbors easily defined
  2. Constant volume of data exchanged between neighbors: $D_c$
  3. Processors virtually organized into a ring
Slicing data

- Data: a 2-D array

- Uni-dimensional partitioning into vertical slices

- Consequences:
  1. Borders and neighbors easily defined
  2. Constant volume of data exchanged between neighbors: $D_c$
  3. Processors virtually organized into a ring
Slicing data

- Data: a 2-D array

- Uni-dimensional partitioning into vertical slices

- Consequences:
  1. Borders and neighbors easily defined
  2. Constant volume of data exchanged between neighbors: $D_c$
  3. Processors virtually organized into a ring
Notations

- **Processors**: $P_1, \ldots, P_p$
- Processor $P_i$ executes a unit task in time $w_i$
- Overall amount of work $D_w$;
  - Share of $P_i$: $\alpha_i.D_w$ processed in time $\alpha_i.D_w.w_i$
  - ($\alpha_i \geq 0$, $\sum_j \alpha_j = 1$)
- Cost of a unit-size communication from $P_i$ to $P_j$: $c_{i,j}$
- Cost of a send from $P_i$ to its successor in the ring: $D_c c_{i,\text{succ}(i)}$
Notations

- Processors: $P_1, \ldots, P_p$
- Processor $P_i$ executes a unit task in time $w_i$
- Overall amount of work $D_w$;
  Share of $P_i$: $\alpha_i.D_w$ processed in time $\alpha_i.D_w.w_i$
  ($\alpha_i \geq 0, \sum_j \alpha_j = 1$)
- Cost of a unit-size communication from $P_i$ to $P_j$: $c_{i,j}$
- Cost of a send from $P_i$ to its successor in the ring: $D_c.c_{i,\text{succ}(i)}$
Notations

- Processors: $P_1, \ldots, P_p$
- Processor $P_i$ executes a unit task in time $w_i$
- Overall amount of work $D_w$;
  Share of $P_i$: $\alpha_i.D_w$ processed in time $\alpha_i.D_w.w_i$
  ($\alpha_i \geq 0$, $\sum_j \alpha_j = 1$)
- Cost of a unit-size communication from $P_i$ to $P_j$: $c_{i,j}$
- Cost of a send from $P_i$ to its successor in the ring: $D_c.c_{i,\text{succ}(i)}$
Notations

- Processors: \( P_1, \ldots, P_p \)
- Processor \( P_i \) executes a unit task in time \( w_i \)
- Overall amount of work \( D_w \);
  Share of \( P_i \): \( \alpha_i.D_w \) processed in time \( \alpha_i.D_w.w_i \)
  \( (\alpha_i \geq 0, \sum_j \alpha_j = 1) \)
- Cost of a unit-size communication from \( P_i \) to \( P_j \): \( c_{i,j} \)
- Cost of a send from \( P_i \) to its successor in the ring: \( D_c.c_{i,\text{succ}(i)} \)
Notations

- Processors: $P_1, \ldots, P_p$
- Processor $P_i$ executes a unit task in time $w_i$
- Overall amount of work $D_w$;
  Share of $P_i$: $\alpha_i.D_w$ processed in time $\alpha_i.D_w.w_i$
  ($\alpha_i \geq 0$, $\sum_j \alpha_j = 1$)

- Cost of a unit-size communication from $P_i$ to $P_j$: $c_{i,j}$
- Cost of a send from $P_i$ to its successor in the ring: $D_{c}.c_{i,\text{succ}(i)}$
Communications: 1-port model

A processor can:

- send at most one message at any time
- receive at most one message at any time
- send and receive a message simultaneously
Objective

1. Select $q$ processors out of $p$ available resources
2. Arrange them along a ring
3. Distribute data

Minimize:

$$\max_{1 \leq i \leq p} \mathbb{I}\{i\}[\alpha_i \cdot D_{w_i} \cdot w_i + D_c \cdot (c_{i,\text{pred}(i)} + c_{i,\text{succ}(i)})]$$

where $\mathbb{I}\{i\}[x] = 1$ if $P_i$ participates in the computation, and 0 otherwise
Objective

1. Select $q$ processors out of $p$ available resources
2. Arrange them along a ring
3. Distribute data

Minimize:

$$\max_{1 \leq i \leq p} \mathbb{I}\{i\}[\alpha_i \cdot D_{w, i} + D_c (c_{i, \text{pred}(i)} + c_{i, \text{succ}(i)})]$$

where $\mathbb{I}\{i\}[x] = 1$ if $P_i$ participates in the computation, and 0 otherwise
Objective

1. Select $q$ processors out of $p$ available resources
2. Arrange them along a ring
3. Distribute data

Minimize:

$$\max_{1 \leq i \leq p} \mathbb{I}\{i\} [\alpha_i D_w w_i + D_c (c_i, \text{pred}(i) + c_i, \text{succ}(i))]$$

where $\mathbb{I}\{i\}[x] = 1$ if $P_i$ participates in the computation, and 0 otherwise
Objective

1. Select $q$ processors out of $p$ available resources
2. Arrange them along a ring
3. Distribute data

Minimize:

$$\max_{1 \leq i \leq p} \mathbb{I}\{i\}[\alpha_i D_w w_i + D_c (c_{i,\text{pred}(i)} + c_{i,\text{succ}(i)})]$$

where $\mathbb{I}\{i\}[x] = 1$ if $P_i$ participates in the computation, and 0 otherwise
Homogeneous fully-connected network

1. There exists a communication link between any processor pair.
2. All links have same capacity.

\[(\forall i, j \ c_{i,j} = c)\]
Either most powerful processor performs all the work, or all processors participate

If all processors participate, all terminate work simultaneously

\((\forall i, \alpha_i.D_{w_i} = \tau, \text{ so } 1 = \sum_i \frac{\tau}{D_{w_i}})\)

Time of optimal solution:

\[ T_{\text{step}} = \min \left\{ D_{w_{\text{min}}}, D_{w} \cdot \frac{1}{\sum_i \frac{1}{w_i}} + 2.D_{c_{\text{c}}} \right\} \]
Results

- Either most powerful processor performs all the work, or all processors participate.
- If all processors participate, all terminate work simultaneously:
  \[
  \forall i, \quad \alpha_i \cdot D_{w.w_i} = \tau, \quad \text{so} \quad 1 = \sum_i \frac{\tau}{D_{w.w_i}}
  \]
- Time of optimal solution:
  \[
  T_{\text{step}} = \min \left\{ D_{w.w_{\text{min}}}, D_w \cdot \frac{1}{\sum_i \frac{1}{w_i}} + 2 \cdot D_c.c \right\}
  \]
Either most powerful processor performs all the work, or all processors participate

If all processors participate, all terminate work simultaneously
\((\forall i, \alpha_i.D_w.w_i = \tau, \text{ so } 1 = \sum_i \frac{\tau}{D_w.w_i})\)

Time of optimal solution:

\[
T_{\text{step}} = \min \left\{ D_w.w_{\text{min}}, D_w \cdot \frac{1}{\sum_i \frac{1}{w_i}} + 2.D_c.c \right\}
\]
Results

- Either most powerful processor performs all the work, or all processors participate
- If all processors participate, all terminate work simultaneously
  \((\forall i, \alpha_i.D_w.w_i = \tau, \text{ so } 1 = \sum_i \frac{\tau}{D_w.w_i})\)
- Time of optimal solution:

\[
T_{\text{step}} = \min \left\{ D_w.w_{\text{min}}, D_w.\frac{1}{\sum_i \frac{1}{w_i}} + 2.D_c.c \right\}
\]
Heterogeneous fully-connected network

1. There exists a communication link between any processor pair
2. Links have different capacities
If all processors participate (1/3)

All processors end simultaneously
If all processors participate (2/3)

- All processors end simultaneously

\[ T_{\text{step}} = \alpha_i \cdot D_w \cdot w_i + D_c \cdot (c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}) \]

\[
\sum_{i=1}^{p} \alpha_i = 1 \implies \sum_{i=1}^{p} \frac{T_{\text{step}} - D_c \cdot (c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)})}{D_w \cdot w_i} = 1
\]

\[
\frac{T_{\text{step}}}{D_w \cdot w_{\text{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}
\]

where \( w_{\text{cumul}} = \frac{1}{\sum_{i} \frac{1}{w_i}} \)
If all processors participate (2/3)

- All processors end simultaneously

\[ T_{\text{step}} = \alpha_i \cdot D_w \cdot w_i + D_c \cdot (c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}) \]

\[
\sum_{i=1}^{p} \alpha_i = 1 \Rightarrow \sum_{i=1}^{p} \frac{T_{\text{step}} - D_c \cdot (c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)})}{D_w \cdot w_i} = 1
\]

\[
\frac{T_{\text{step}}}{D_w \cdot w_{\text{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}
\]

where \( w_{\text{cumul}} = \frac{1}{\sum_i \frac{1}{w_i}} \)
If all processors participate (3/3)

\[ \frac{T_{\text{step}}}{D_w \cdot w_{\text{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i} \]

\[ T_{\text{step}} \text{ minimal } \iff \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i} \text{ is minimal} \]

Search an hamiltonian cycle of minimal weight in a graph where the edge from \( P_i \) to \( P_j \) has a weight of \( d_{i,j} = \frac{c_{i,j}}{w_i} + \frac{c_{j,i}}{w_j} \)

NP-complete problem
If all processors participate (3/3)

\[
\frac{T_{\text{step}}}{D_w w_{\text{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}
\]

\[T_{\text{step}} \text{ minimal } \iff \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i} \text{ is minimal}\]

Search an hamiltonian cycle of minimal weight in a graph where the edge from \(P_i\) to \(P_j\) has a weight of \(d_{i,j} = \frac{c_{i,j}}{w_i} + \frac{c_{j,i}}{w_j}\)

NP-complete problem
If all processors participate (3/3)

\[
\frac{T_{\text{step}}}{D_w \cdot w_{\text{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}
\]

\[T_{\text{step}} \text{ minimal } \iff \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i} \text{ is minimal}\]

Search an hamiltonian cycle of minimal weight in a graph where the edge from \(P_i\) to \(P_j\) has a weight of

\[d_{i,j} = \frac{c_{i,j}}{w_i} + \frac{c_{j,i}}{w_j}\]

NP-complete problem
If all processors participate (3/3)

\[
\frac{T_{\text{step}}}{D_w \cdot W_{\text{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}
\]

\[T_{\text{step}} \text{ minimal } \Leftrightarrow \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i} \text{ is minimal}\]

Search an hamiltonian cycle of minimal weight in a graph where the edge from \(P_i\) to \(P_j\) has a weight of \(d_{i,j} = \frac{c_{i,j}}{w_i} + \frac{c_{j,i}}{w_j}\)

NP-complete problem
If all processors participate: linear program

\[ \text{MINIMIZE} \quad \sum_{i=1}^{p} \sum_{j=1}^{p} d_{i,j} x_{i,j}, \]

\[ \text{SATISFYING THE (IN)EQUATIONS} \]

\[
\begin{align*}
& (1) \quad \sum_{j=1}^{p} x_{i,j} = 1 \quad 1 \leq i \leq p \\
& (2) \quad \sum_{i=1}^{p} x_{i,j} = 1 \quad 1 \leq j \leq p \\
& (3) \quad x_{i,j} \in \{0, 1\} \quad 1 \leq i, j \leq p \\
& (4) \quad u_i - u_j + p \cdot x_{i,j} \leq p - 1 \quad 2 \leq i, j \leq p, i \neq j \\
& (5) \quad u_i \text{ integer, } u_i \geq 0 \quad 2 \leq i \leq p \\
\end{align*}
\]

\[ x_{i,j} = 1 \text{ if, and only if, the edge from } P_i \text{ to } P_j \text{ is used} \]
General case: linear program

Best ring made of $q$ processors

**Minimize** $T$ satisfying the (in)equations

\begin{align*}
(1) & \quad x_{i,j} \in \{0, 1\} & 1 \leq i, j \leq p \\
(2) & \quad \sum_{i=1}^{p} x_{i,j} \leq 1 & 1 \leq j \leq p \\
(3) & \quad \sum_{i=1}^{p} \sum_{j=1}^{p} x_{i,j} = q \\
(4) & \quad \sum_{i=1}^{p} x_{i,j} = \sum_{i=1}^{p} x_{j,i} & 1 \leq j \leq p \\
(5) & \quad \sum_{i=1}^{p} \alpha_i = 1 \\
(6) & \quad \alpha_i \leq \sum_{j=1}^{p} x_{i,j} & 1 \leq i \leq p \\
(7) & \quad \alpha_i \cdot w_i + \frac{D_c}{D_w} \sum_{j=1}^{p} (x_{i,j} c_{i,j} + x_{j,i} c_{j,i}) \leq T & 1 \leq i \leq p \\
(8) & \quad \sum_{i=1}^{p} y_i = 1 \\
(9) & \quad -p \cdot y_i - p \cdot y_j + u_i - u_j + q \cdot x_{i,j} \leq q - 1 & 1 \leq i, j \leq p, i \neq j \\
(10) & \quad y_i \in \{0, 1\} & 1 \leq i \leq p \\
(11) & \quad u_i \text{ integer, } u_i \geq 0 & 1 \leq i \leq p
\end{align*}
Linear programming

- Problems with rational variables: can be solved in polynomial time (in the size of the problem)
- Problems with integer variables: solved in exponential time in the worst case
- No relaxation in rational numbers seems possible here...
Linear programming

- Problems with rational variables: can be solved in polynomial time (in the size of the problem)
- Problems with integer variables: solved in exponential time in the worst case
- No relaxation in rational numbers seems possible here...
Linear programming

- Problems with rational variables: can be solved in polynomial time (in the size of the problem)
- Problems with integer variables: solved in exponential time in the worst case
- No relaxation in rational numbers seems possible here...
And, in practice?

**If all processors participate.** Use a heuristic to solve the traveling salesman problem (as Lin-Kernighan)

No guarantee, but excellent results in practice.

**General case.**

1. Exhaustive search: feasible up to a dozen of processors
2. Greedy heuristic:
   - initially take best pair of processors
   - for a given ring, try to insert any unused processor in between any pair of neighbor processors in the ring
And, in practice?

If all processors participate. Use a heuristic to solve the traveling salesman problem (as Lin-Kernighan)
No guarantee, but excellent results in practice.

General case.

1. Exhaustive search: feasible up to a dozen of processors
2. Greedy heuristic:
   - initially take best pair of processors
   - for a given ring, try to insert any unused processor in between any pair of neighbor processors in the ring
And, in practice?

If all processors participate. Use a heuristic to solve the traveling salesman problem (as Lin-Kernighan)
No guarantee, but excellent results in practice.

General case.

1. Exhaustive search: feasible up to a dozen of processors
2. Greedy heuristic:
   - initially take best pair of processors
   - for a given ring, try to insert any unused processor in between any pair of neighbor processors in the ring
And, in practice?

**If all processors participate.** Use a heuristic to solve the traveling salesman problem (as Lin-Kernighan)
No guarantee, but excellent results in practice.

**General case.**

1. **Exhaustive search:** feasible up to a dozen of processors
2. **Greedy heuristic:**
   - initially take best pair of processors
   - for a given ring, try to insert any unused processor in between any pair of neighbor processors in the ring
And, in practice?

**If all processors participate.** Use a heuristic to solve the traveling salesman problem (as Lin-Kernighan)
No guarantee, but excellent results in practice.

**General case.**

1. Exhaustive search: feasible up to a dozen of processors
2. Greedy heuristic:
   - initially take best pair of processors
   - for a given ring, try to insert any unused processor in between any pair of neighbor processors in the ring
Heterogeneous network (general case)

Heterogeneous platform

Virtual ring

Take link sharing into account
Heterogeneous network (general case)

Heterogeneous platform

Virtual ring

Take link sharing into account
Heterogeneous network (general case)

Heterogeneous platform

Virtual ring

Take link sharing into account
New notations

- **Set of communications links:** $e_1, \ldots, e_n$
- **Bandwidth of link** $e_m$: $b_{e_m}$
- **There is a path** $S_i$ **from** $P_i$ **to** $P_{\text{succ}(i)}$ **in the network**
  - $S_i$ **uses a fraction** $s_{i,m}$ **of the bandwidth** $b_{e_m}$ **of link** $e_m$
  - $P_i$ **needs a time** $D_c \cdot \frac{1}{\min_{e_m \in S_i} s_{i,m}}$ **to send a message of size** $D_c$
    **to its successor**
  - **Constraints on the bandwidth of** $e_m$: $\sum_{1 \leq i \leq p} s_{i,m} \leq b_{e_m}$
- **Symmetrically,** there is a path $P_i$ from $P_i$ to $P_{\text{pred}(i)}$ in the network, which uses a fraction $p_{i,m}$ of the bandwidth $b_{e_m}$ of link $e_m$
New notations

- **Set of communications links:** $e_1, \ldots, e_n$
- **Bandwidth of link** $e_m$: $b_{e_m}$

There is a path $S_i$ from $P_i$ to $P_{\text{succ}(i)}$ in the network:

- $S_i$ uses a fraction $s_{i,m}$ of the bandwidth $b_{e_m}$ of link $e_m$
- $P_i$ needs a time $D_c \cdot \frac{1}{\min_{e_m \in S_i} s_{i,m}}$ to send a message of size $D_c$ to its successor
- Constraints on the bandwidth of $e_m$: $\sum_{1 \leq i \leq p} s_{i,m} \leq b_{e_m}$

Symmetrically, there is a path $P_i$ from $P_i$ to $P_{\text{pred}(i)}$ in the network, which uses a fraction $p_{i,m}$ of the bandwidth $b_{e_m}$ of link $e_m$
New notations

- Set of communications links: $e_1, \ldots, e_n$
- Bandwidth of link $e_m$: $b_{em}$
- There is a path $S_i$ from $P_i$ to $P_{\text{succ}(i)}$ in the network
  - $S_i$ uses a fraction $s_{i,m}$ of the bandwidth $b_{em}$ of link $e_m$
  - $P_i$ needs a time $D_c \cdot \frac{1}{\min_{e_m \in S_i} s_{i,m}}$ to send a message of size $D_c$
    to its successor
  - Constraints on the bandwidth of $e_m$: $\sum_{1 \leq i \leq p} s_{i,m} \leq b_{em}$
- Symmetrically, there is a path $P_i$ from $P_i$ to $P_{\text{pred}(i)}$ in the network, which uses a fraction $p_{i,m}$ of the bandwidth $b_{em}$ of link $e_m$
New notations

- Set of communications links: $e_1, \ldots, e_n$
- Bandwidth of link $e_m$: $b_{em}$
- There is a path $S_i$ from $P_i$ to $P_{\text{succ}(i)}$ in the network
  - $S_i$ uses a fraction $s_{i,m}$ of the bandwidth $b_{em}$ of link $e_m$
  - $P_i$ needs a time $D_c \cdot \frac{1}{\min_{e_m \in S_i} s_{i,m}}$ to send a message of size $D_c$ to its successor
  - Constraints on the bandwidth of $e_m$: $\sum_{1 \leq i \leq p} s_{i,m} \leq b_{em}$
- Symmetrically, there is a path $P_i$ from $P_i$ to $P_{\text{pred}(i)}$ in the network, which uses a fraction $p_{i,m}$ of the bandwidth $b_{em}$ of link $e_m$
New notations

- Set of communications links: \( e_1, \ldots, e_n \)
- Bandwidth of link \( e_m \): \( b_{e_m} \)
- There is a path \( S_i \) from \( P_i \) to \( P_{\text{succ}(i)} \) in the network
  - \( S_i \) uses a fraction \( s_{i,m} \) of the bandwidth \( b_{e_m} \) of link \( e_m \)
  - \( P_i \) needs a time \( D_c \cdot \frac{1}{\min_{e_m \in S_i} s_{i,m}} \) to send a message of size \( D_c \) to its successor
  - Constraints on the bandwidth of \( e_m \): \( \sum_{1 \leq i \leq p} s_{i,m} \leq b_{e_m} \)
- Symmetrically, there is a path \( P_i \) from \( P_i \) to \( P_{\text{pred}(i)} \) in the network, which uses a fraction \( p_{i,m} \) of the bandwidth \( b_{e_m} \) of link \( e_m \)
New notations

- Set of communications links: $e_1, ..., e_n$
- Bandwidth of link $e_m$: $b_e^m$
- There is a path $S_i$ from $P_i$ to $P_{\text{succ}(i)}$ in the network
  - $S_i$ uses a fraction $s_{i,m}$ of the bandwidth $b_e^m$ of link $e_m$
  - $P_i$ needs a time $D_c \cdot \frac{1}{\min_{e_m \in S_i} s_{i,m}}$ to send a message of size $D_c$ to its successor
  - Constraints on the bandwidth of $e_m$: $\sum_{1 \leq i \leq p} s_{i,m} \leq b_e^m$
- Symmetrically, there is a path $P_i$ from $P_i$ to $P_{\text{pred}(i)}$ in the network, which uses a fraction $p_{i,m}$ of the bandwidth $b_e^m$ of link $e_m$
New notations

- Set of communications links: $e_1, \ldots, e_n$
- Bandwidth of link $e_m$: $b_{e_m}$
- There is a path $S_i$ from $P_i$ to $P_{\text{succ}(i)}$ in the network
  - $S_i$ uses a fraction $s_{i,m}$ of the bandwidth $b_{e_m}$ of link $e_m$
  - $P_i$ needs a time $D_c \cdot \frac{1}{\min_{e_m \in S_i} s_{i,m}}$ to send a message of size $D_c$ to its successor
  - Constraints on the bandwidth of $e_m$: $\sum_{1 \leq i \leq p} s_{i,m} \leq b_{e_m}$
- Symmetrically, there is a path $P_i$ from $P_i$ to $P_{\text{pred}(i)}$ in the network, which uses a fraction $p_{i,m}$ of the bandwidth $b_{e_m}$ of link $e_m$
Toy example: choosing the ring

7 processors and 8 bidirectional communications links

We choose a ring of 5 processors:

$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5$ (we use neither $Q$, nor $R$)
Toy example: choosing the ring

- 7 processors and 8 bidirectional communications links
- We choose a ring of 5 processors:
  \[ P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \] (we use neither \( Q \), nor \( R \))
Toy example: choosing routing paths

From $P_1$ to $P_2$, use links $a$ and $b$: $S_1 = \{a, b\}$.

From $P_2$ to $P_1$, use links $b$, $g$ and $h$: $\mathcal{P}_2 = \{b, g, h\}$.

From $P_1$: to $P_2$, $S_1 = \{a, b\}$ and to $P_5$, $\mathcal{P}_1 = \{h\}$

From $P_2$: to $P_3$, $S_2 = \{c, d\}$ and to $P_1$, $\mathcal{P}_2 = \{b, g, h\}$

From $P_3$: to $P_4$, $S_3 = \{d, e\}$ and to $P_2$, $\mathcal{P}_3 = \{d, e, f\}$

From $P_4$: to $P_5$, $S_4 = \{f, b, g\}$ and to $P_3$, $\mathcal{P}_4 = \{e, d\}$

From $P_5$: to $P_1$, $S_5 = \{h\}$ and to $P_4$, $\mathcal{P}_5 = \{g, b, f\}$
Toy example: choosing routing paths

From $P_1$ to $P_2$, use links $a$ and $b$: $S_1 = \{a, b\}$.
From $P_2$ to $P_1$, use links $b$, $g$ and $h$: $P_2 = \{b, g, h\}$.

From $P_1$: to $P_2$, $S_1 = \{a, b\}$ and to $P_5$, $P_1 = \{h\}$
From $P_2$: to $P_3$, $S_2 = \{c, d\}$ and to $P_1$, $P_2 = \{b, g, h\}$
From $P_3$: to $P_4$, $S_3 = \{d, e\}$ and to $P_2$, $P_3 = \{d, e, f\}$
From $P_4$: to $P_5$, $S_4 = \{f, b, g\}$ and to $P_3$, $P_4 = \{e, d\}$
From $P_5$: to $P_1$, $S_5 = \{h\}$ and to $P_4$, $P_5 = \{g, b, f\}$
Toy example: choosing routing paths

From $P_1$ to $P_2$, use links $a$ and $b$: $S_1 = \{a, b\}$.
From $P_2$ to $P_1$, use links $b$, $g$ and $h$: $P_2 = \{b, g, h\}$.

From $P_1$ to $P_2$, $S_1 = \{a, b\}$ and to $P_5$, $P_1 = \{h\}$
From $P_2$ to $P_3$, $S_2 = \{c, d\}$ and to $P_1$, $P_2 = \{b, g, h\}$
From $P_3$ to $P_4$, $S_3 = \{d, e\}$ and to $P_2$, $P_3 = \{d, e, f\}$
From $P_4$ to $P_5$, $S_4 = \{f, b, g\}$ and to $P_3$, $P_4 = \{e, d\}$
From $P_5$ to $P_1$, $S_5 = \{h\}$ and to $P_4$, $P_5 = \{g, b, f\}$
Toy example: choosing routing paths

From $P_1$ to $P_2$, use links $a$ and $b$: $S_1 = \{a, b\}$.
From $P_2$ to $P_1$, use links $b$, $g$ and $h$: $P_2 = \{b, g, h\}$.

From $P_1$: to $P_2$, $S_1 = \{a, b\}$ and to $P_5$, $P_1 = \{h\}$
From $P_2$: to $P_3$, $S_2 = \{c, d\}$ and to $P_1$, $P_2 = \{b, g, h\}$
From $P_3$: to $P_4$, $S_3 = \{d, e\}$ and to $P_2$, $P_3 = \{d, e, f\}$
From $P_4$: to $P_5$, $S_4 = \{f, b, g\}$ and to $P_3$, $P_4 = \{e, d\}$
From $P_5$: to $P_1$, $S_5 = \{h\}$ and to $P_4$, $P_5 = \{g, b, f\}$
Toy example: bandwidth sharing

From $P_1$ to $P_2$ we use links $a$ and $b$: $c_{1,2} = \frac{1}{\min(s_{1,a}, s_{1,b})}$.

From $P_1$ to $P_5$ we use link $h$: $c_{1,5} = \frac{1}{p_{1,h}}$.

Set of all sharing constraints:

- Lien $a$: $s_{1,a} \leq b_a$
- Lien $b$: $s_{1,b} + s_{4,b} + p_{2,b} + p_{5,b} \leq b_b$
- Lien $c$: $s_{2,c} \leq b_c$
- Lien $d$: $s_{2,d} + s_{3,d} + p_{3,d} + p_{4,d} \leq b_d$
- Lien $e$: $s_{3,e} + p_{3,e} + p_{4,e} \leq b_e$
- Lien $f$: $s_{4,f} + p_{3,f} + p_{5,f} \leq b_f$
- Lien $g$: $s_{4,g} + p_{2,g} + p_{5,g} \leq b_g$
- Lien $h$: $s_{5,h} + p_{1,h} + p_{2,h} \leq b_h$
Toy example: bandwidth sharing

From $P_1$ to $P_2$ we use links $a$ and $b$: $c_{1,2} = \frac{1}{\min(s_{1,a},s_{1,b})}$.

From $P_1$ to $P_5$ we use link $h$: $c_{1,5} = \frac{1}{p_{1,h}}$.

Set of all sharing constraints:

Lien $a$: $s_{1,a} \leq b_a$
Lien $b$: $s_{1,b} + s_{4,b} + p_{2,b} + p_{5,b} \leq b_b$
Lien $c$: $s_{2,c} \leq b_c$
Lien $d$: $s_{2,d} + s_{3,d} + p_{3,d} + p_{4,d} \leq b_d$
Lien $e$: $s_{3,e} + p_{3,e} + p_{4,e} \leq b_e$
Lien $f$: $s_{4,f} + p_{3,f} + p_{5,f} \leq b_f$
Lien $g$: $s_{4,g} + p_{2,g} + p_{5,g} \leq b_g$
Lien $h$: $s_{5,h} + p_{1,h} + p_{2,h} \leq b_h$
Toy example: final quadratic system

Minimize \[ \max_{1 \leq i \leq 5} \left( \alpha_i D_w w_i + D_c (c_{i-1} + c_{i+1}) \right) \]

under the constraints

\[
\begin{align*}
\sum_{i=1}^{5} \alpha_i &= 1 \\
s_{1,a} &\leq b_a \\
s_{2,d} + s_{3,d} + p_{3,d} + p_{4,d} &\leq b_d \\
s_{4,g} + p_{2,g} + p_{5,g} &\leq b_g \\
s_{1,a} c_{1,2} &\geq 1 \\
s_{2,c} c_{2,3} &\geq 1 \\
p_{2,g} c_{2,1} &\geq 1 \\
s_{3,e} c_{3,4} &\geq 1 \\
p_{3,f} c_{3,2} &\geq 1 \\
s_{4,g} c_{4,5} &\geq 1 \\
s_{5,h} c_{5,1} &\geq 1 \\
p_{5,f} c_{5,4} &\geq 1 \\
s_{1,b} + s_{4,b} + p_{2,b} + p_{5,b} &\leq b_b \\
s_{3,e} + p_{3,e} + p_{4,e} &\leq b_e \\
s_{5,h} + p_{1,h} + p_{2,h} &\leq b_h \\
p_{1,h} c_{1,5} &\geq 1 \\
p_{2,b} c_{2,1} &\geq 1 \\
p_{2,h} c_{2,1} &\geq 1 \\
p_{3,d} c_{3,4} &\geq 1 \\
p_{3,d} c_{3,2} &\geq 1 \\
s_{4,f} c_{4,5} &\geq 1 \\
p_{4,e} c_{4,3} &\geq 1 \\
p_{5,g} c_{5,4} &\geq 1 \\
p_{4,d} c_{4,3} &\geq 1 \\
p_{5,b} c_{5,4} &\geq 1
\end{align*}
\]
Toy example: the moral

Problem sums up to a quadratic system if

1. processors are already selected
2. processors are already ordered into a ring
3. communication paths are already known

In other words: a quadratic system if the ring is known.

If the ring is known:
- Complete graph: closed-form expression
- General graph: quadratic system

Is the more complex network model with link contention worth the trouble?
Toy example: the moral

Problem sums up to a quadratic system if

1. processors are already selected
2. processors are already ordered into a ring
3. communication paths are already known

In other words: a quadratic system if the ring is known.

If the ring is known:

- Complete graph: closed-form expression
- General graph: quadratic system

Is the more complex network model \textit{with link contention} worth the trouble?
Toy example: the moral

Problem sums up to a quadratic system if

1. processors are already selected
2. processors are already ordered into a ring
3. communication paths are already known

In other words: a quadratic system if the ring is known.

If the ring is known:

- Complete graph: closed-form expression
- General graph: quadratic system

Is the more complex network model with link contention worth the trouble?
Toy example: the moral

Problem sums up to a quadratic system if

1. processors are already selected
2. processors are already ordered into a ring
3. communication paths are already known

In other words: a quadratic system if the ring is known.

If the ring is known:

- Complete graph: closed-form expression
- General graph: quadratic system

Is the more complex network model with link contention worth the trouble?
Toy example: the moral

Problem sums up to a quadratic system if

1. processors are already selected
2. processors are already ordered into a ring
3. communication paths are already known

In other words: a quadratic system if the ring is known.

If the ring is known:

- Complete graph: closed-form expression
- General graph: quadratic system

Is the more complex network model with link contention worth the trouble?
Toy example: the moral

Problem sums up to a quadratic system if

1. processors are already selected
2. processors are already ordered into a ring
3. communication paths are already known

In other words: a quadratic system if the ring is known.

If the ring is known:

- Complete graph: closed-form expression
- General graph: quadratic system

Is the more complex network model with link contention worth the trouble?
And, in practice?

Adapt greedy heuristic:

1. Initially: best processor pair
2. For each processor $P_k$ (not already included in the ring)
   - For each pair $(P_i, P_j)$ of neighbors in the ring
     1. Build graph of unused bandwidths
        (without considering the paths between $P_i$ and $P_j$)
     2. Compute shortest paths (in terms of bandwidth) between $P_k$ and $P_i$ and $P_j$
     3. Evaluate solution

3. Keep best solution found at step 2 and start again
   + refinements (*max-min fairness*, quadratic solving)
Is this meaningful?

- No guarantee, neither theoretical, nor practical

Simple solution:
1. build complete graph whose edges are labeled with bandwidths of best communication paths
2. apply the heuristic for complete graphs
3. allocate bandwidths
Is this meaningful?

- No guarantee, neither theoretical, nor practical
- Simple solution:
  1. build complete graph whose edges are labeled with bandwidths of best communication paths
  2. apply the heuristic for complete graphs
  3. allocate bandwidths
Is this meaningful?

- No guarantee, neither theoretical, nor practical
- Simple solution:
  1. build complete graph whose edges are labeled with bandwidths of best communication paths
  2. apply the heuristic for complete graphs
  3. allocate bandwidths
Is this meaningful?

• No guarantee, neither theoretical, nor practical

• Simple solution:
  1. build complete graph whose edges are labeled with bandwidths of best communication paths
  2. apply the heuristic for complete graphs
  3. allocate bandwidths
Is this meaningful?

- No guarantee, neither theoretical, nor practical
- Simple solution:
  1. Build complete graph whose edges are labeled with bandwidths of best communication paths
  2. Apply the heuristic for complete graphs
  3. Allocate bandwidths
An example of an actual platform (Lyon)

Topology

<table>
<thead>
<tr>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_5$</th>
<th>$P_6$</th>
<th>$P_7$</th>
<th>$P_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0206</td>
<td>0.0206</td>
<td>0.0206</td>
<td>0.0206</td>
<td>0.0291</td>
<td>0.0206</td>
<td>0.0087</td>
<td>0.0206</td>
<td>0.0206</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P_9$</th>
<th>$P_{10}$</th>
<th>$P_{11}$</th>
<th>$P_{12}$</th>
<th>$P_{13}$</th>
<th>$P_{14}$</th>
<th>$P_{15}$</th>
<th>$P_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0206</td>
<td>0.0206</td>
<td>0.0206</td>
<td>0.0291</td>
<td>0.0451</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Processors processing times (in seconds par megaflop)
Abstracting the Lyon platform.
Results

First heuristic building the ring without taking link sharing into account

Second heuristic taking link sharing into account (and with quadratic programming)

<table>
<thead>
<tr>
<th>Ratio $D_c/D_w$</th>
<th>H1</th>
<th>H2</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.64</td>
<td>0.008738 (1)</td>
<td>0.008738 (1)</td>
<td>0%</td>
</tr>
<tr>
<td>0.064</td>
<td>0.018837 (13)</td>
<td>0.006639 (14)</td>
<td>64.75%</td>
</tr>
<tr>
<td>0.0064</td>
<td>0.003819 (13)</td>
<td>0.001975 (14)</td>
<td>48.28%</td>
</tr>
</tbody>
</table>

Table: $T_{step}/D_w$ for each heuristic on the Lyon and Strasbourg platforms (numbers in parentheses show size of solution rings)
And with non dedicated platforms?

Available processing power of each processor changes over time

Available bandwidth of each communication link changes over time

⇒ Need to reconsider current allocation

⇒ Introduce (dynamic) redistribution algorithms
A possible approach

- If actual performance “too much” different from expected characteristics when building solution
  
  Actual criterion defining “too much”?  

- If actual performance “very” different
  
  - compute a new ring
  
  - redistribute data from old ring to new one

  Actual criterion defining “very”?  
  Cost of the redistribution?  

- If the actual performance is “a little” different
  
  - compute new load-balancing in existing ring
  
  - redistribute data in existing ring

  How to efficiently do the redistribution?
A possible approach

- If actual performance “too much” different from expected characteristics when building solution
  
  **Actual criterion defining “too much”?**

- If actual performance “very” different
  - compute a new ring
  - redistribute data from old ring to new one
  
  **Actual criterion defining “very”?**
  
  **Cost of the redistribution?**

- If the actual performance is “a little” different
  - compute new load-balancing in existing ring
  - redistribute data in existing ring
  
  **How to efficiently do the redistribution?**
Load-balancing

**Principle:** ring is modified only if this is profitable

- $T_{\text{step}}$: length of an iteration *before* load-balancing
- $T'_{\text{step}}$: length of an iteration *after* load-balancing
- $T_{\text{redistribution}}$: redistribution cost
- $n_{\text{iter}}$: number of remaining iterations

**Condition:**

$$T_{\text{redistribution}} + n_{\text{iter}} \times T'_{\text{step}} \leq n_{\text{iter}} \times T_{\text{step}}$$

**Redistribution algorithms** for homo/hetero uni/bi-dir rings

(Well, let’s do this another time . . . )
Load-balancing

**Principle:** ring is modified only if this is profitable

- \( T_{\text{step}} \): length of an iteration *before* load-balancing
- \( T'_{\text{step}} \): length of an iteration *after* load-balancing
- \( T_{\text{redistribution}} \): redistribution cost
- \( n_{\text{iter}} \): number of remaining iterations

**Condition:**

\[
T_{\text{redistribution}} + n_{\text{iter}} \times T'_{\text{step}} \leq n_{\text{iter}} \times T_{\text{step}}
\]

**Redistribution algorithms** for homo/hetero uni/bi-dir rings

(Well, let’s do this another time . . .)
Lesson learnt?

Realistic networks models: mandatory but less tractable

... Need find good trade-offs.

Would be even more complicated with hierarchical architectures 😞
Lesson learnt?

Realistic networks models: mandatory but less tractable

... Need find good trade-offs.

Would be even more complicated with hierarchical architectures ☹️
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion
Outline

1 Parallel algorithms
   • Independent tasks
   • A simple tiling problem
   • Matrix product (ScaLAPACK)
   • Matrix product (master-slave)
   • Iterative algorithms

2 Scheduling
   • Background: scheduling DAGs
   • Packet routing
   • Steady-state scheduling
   • Multiple applications

3 Pipeline workflows

4 Models and real life

5 Conclusion
Traditional scheduling – Framework

- **Application** = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - $E$ = dependence constraints
  - $w(T)$ = computational cost of task $T$ (execution time)
  - $c(T, T')$ = communication cost (data sent from $T$ to $T'$)

- **Platform**
  - Set of $p$ identical processors

- **Schedule**
  - $\sigma(T)$ = date to begin execution of task $T$
  - $\text{alloc}(T)$ = processor assigned to it
Traditional scheduling – Constraints

- **Data dependences** If \((T, T') \in E\) then
  - if \(\text{alloc}(T) = \text{alloc}(T')\) then \(\sigma(T) + w(T) \leq \sigma(T')\)
  - if \(\text{alloc}(T) \neq \text{alloc}(T')\) then \(\sigma(T) + w(T) + c(T, T') \leq \sigma(T')\)

- **Resource constraints**

\[
\text{alloc}(T) = \text{alloc}(T') \Rightarrow (\sigma(T) + w(T) \leq \sigma(T')) \text{ or } (\sigma(T') + w(T') \leq \sigma(T))
\]
Introduction

Parallel algorithms

Scheduling

Pipeline workflows

Models and real life

Conclusion

Traditional scheduling – Objective functions

- **Makespan** or total execution time

\[ MS(\sigma) = \max_{T \in T} (\sigma(T) + w(T)) \]

- Other classical objectives:
  - Sum of completion times
  - With release dates: max flow (response time), or sum flow
  - Fairness oriented: max stretch, or sum stretch
Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step

- **Very crude** for network resources
  - Unlimited number of simultaneous sends/receives per processor
  - No contention → unbounded bandwidth on any link
  - Fully connected interconnection graph (clique)

- In fact, model assumes *infinite* network capacity
Makespan minimization

- **NP-hardness**
  - $Pb(p)$ NP-complete for independent tasks and no communications
    - $(E = \emptyset, p = 2$ and $c = 0)$
  - $Pb(p)$ NP-complete for UET-UCT graphs $(w = c = 1)$

- **Approximation algorithms**
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no $\lambda$-approximation in general
List scheduling – Without communications

*Initialization*:
- Compute priority level of all tasks
- Priority queue = list of free tasks (tasks without predecessors) sorted by priority

*While there remain tasks to execute*:
- Add new free tasks, if any, to the queue.
- If there are $q$ available processors and $r$ tasks in the queue, remove first $\min(q, r)$ tasks from the queue and execute them

*Priority level*
- Use critical path: longest path from the task to an exit node
- Computed recursively by a bottom-up traversal of the graph
List scheduling – Without communications

Initialization:
- Compute priority level of all tasks
- Priority queue = list of free tasks (tasks without predecessors) sorted by priority

While there remain tasks to execute:
- Add new free tasks, if any, to the queue.
- If there are $q$ available processors and $r$ tasks in the queue, remove first $\min(q, r)$ tasks from the queue and execute them

Priority level
- Use critical path: longest path from the task to an exit node
- Computed recursively by a bottom-up traversal of the graph
List scheduling – With communications (1/2)

- **Priority level**
  - Use *pessimistic* critical path: include all edge costs in the weight
  - Computed recursively by a bottom-up traversal of the graph

- **MCP *Modified Critical Path***
  - Assign free task with highest priority to *best* processor
  - Best processor = finishes execution first, given already taken scheduling decisions
  - Free tasks may not be ready for execution (communication delays)
  - May explore inserting the task in empty slots of schedule
  - Complexity $O(|V| \log |V| + (|E| + |V|)p)$
List scheduling – With communications (2/2)

- EFT *Earliest Finish Time*
  - Dynamically recompute priorities of free tasks
  - Select free task that finishes execution first (on best processor), given already taken scheduling decisions
  - Higher complexity $O(|V|^3 p)$
  - May miss “urgent” tasks on critical path

- Other approaches
  - Two-step: clustering + load balancing
    - DSC Dominant Sequence Clustering $O((|V| + |E|) \log |V|)$
    - LLB List-based Load Balancing $O(C \log C + |V|)$ ($C$ number of clusters generated by DSC)
  - Low-cost: FCP Fast Critical Path
    - Maintain constant-size sorted list of free tasks:
    - Best processor = first idle or the one sending last message
    - Low complexity $O(|V| \log p + |E|)$
Extending the model to heterogeneous clusters

- Task graph with \( n \) tasks \( T_1, \ldots, T_n \).
- Platform with \( p \) heterogeneous processors \( P_1, \ldots, P_p \).
- Computation costs:
  - \( w_{iq} \) = execution time of \( T_i \) on \( P_q \)
  - \( \overline{w_i} = \frac{\sum_{q=1}^{p} w_{iq}}{p} \) average execution time of \( T_i \)
  - particular case: consistent tasks \( w_{iq} = w_i \times \gamma_q \)
- Communication costs:
  - \( \text{data}(i, j) \): data volume for edge \( e_{ij}: T_i \rightarrow T_j \)
  - \( v_{qr} \): communication time for unit-size message from \( P_q \) to \( P_r \) (zero if \( q = r \))
  - \( \text{com}(i, j, q, r) = \text{data}(i, j) \times v_{qr} \) communication time from \( T_i \) executed on \( P_q \) to \( P_j \) executed on \( P_r \)
  - \( \overline{\text{com}_{ij}} = \text{data}(i, j) \times \frac{\sum_{1 \leq q, r \leq p, q \neq r} v_{qr}}{p(p-1)} \) average communication cost for edge \( e_{ij}: T_i \rightarrow T_j \)
Rewriting constraints

**Dependences** For $e_{ij} : T_i \rightarrow T_j$, $q = \text{alloc}(T_i)$ and $r = \text{alloc}(T_j)$:

$$\sigma(T_i) + w_{iq} + \text{com}(i, j, q, r) \leq \sigma(T_j)$$

**Resources** If $q = \text{alloc}(T_i) = \text{alloc}(T_j)$, then

$$(\sigma(T_i) + w_{iq} \leq \sigma(T_j)) \text{ or } (\sigma(T_j) + w_{jq} \leq \sigma(T_i))$$

**Makespan**

$$\max_{1 \leq i \leq n} \left( \sigma(T_i) + w_{i,\text{alloc}(T_i)} \right)$$
HEFT: Heterogeneous Earliest Finish Time

**Priority level:**
- \( \text{rank}(T_i) = w_i + \max_{T_j \in \text{Succ}(T_i)} (\text{com}_{ij} + \text{rank}(T_j)) \),
- where \( \text{Succ}(T) \) is the set of successors of \( T \)
- Recursive computation by bottom-up traversal of the graph

**Allocation**
- For current task \( T_i \), determine best processor \( P_q \):
  - minimize \( \sigma(T_i) + w_{iq} \)
  - Enforce constraints related to communication costs
  - Insertion scheduling: look for \( t = \sigma(T_i) \) s.t. \( P_q \) is available during interval \( [t, t + w_{iq}] \)

**Complexity:** same as MCP without/with insertion
What's wrong?

- 😊 Nothing (still may need to map a DAG onto a platform!)
- 😞 Absurd communication model:
  - complicated: many parameters to instantiate
  - while not realistic (clique + no contention)
- 😞 Wrong metric: need to relax makespan minimization objective
What’s wrong?

- 😊 Nothing (still may need to map a DAG onto a platform!)
- 😞 Absurd communication model:
  complicated: many parameters to instantiate
  \textit{while not realistic} (clique + no contention)
- 😞 Wrong metric: need to relax makespan minimization objective
What’s wrong?

- 😊 Nothing (still may need to map a DAG onto a platform!)
- 😞 Absurd communication model:
  complicated: many parameters to instantiate
  while not realistic (clique + no contention)
- 😞 Wrong metric: need to relax makespan minimization objective
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion
Routing sets of messages from sources to destinations
Paths not fixed a priori
Packets of same message may follow different paths
Hypotheses

- A packet crosses an edge within one time-step
- At any time-step, at most one packet crosses an edge
Hypotheses

- A packet crosses an edge within one time-step
- At any time-step, at most one packet crosses an edge

**Scheduling**: for each time-step, decide which packet crosses any given edge
\begin{itemize}
  \item \( n^{k,l} \): total number of packets to be routed from \( k \) to \( l \)
  \item \( n_{i,j}^{k,l} \): total number of packets routed from \( k \) to \( l \) and crossing edge \((i,j)\)
\end{itemize}
**Lower bound**

**Congestion** $C_{i,j}$ of edge $(i, j)$

$= \text{total number of packets that cross } (i, j)$

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n_{i,j}^{k,l} \quad C_{\text{max}} = \max_{i,j} C_{i,j}$$

$C_{\text{max}}$ lower bound on schedule makespan

$C^* \geq C_{\text{max}}$

$\Rightarrow$ "Fluidified" solution in $C_{\text{max}}$?
Lower bound

**Congestion** $C_{i,j}$ of edge $(i, j)$

= total number of packets that cross $(i, j)$

$$C_{i,j} = \sum_{(k,l) | n_{k,l} > 0} n_{k,l}$$

$C_{\text{max}} = \max_{i,j} C_{i,j}$

$C_{\text{max}}$ lower bound on schedule makespan

$$C^* \geq C_{\text{max}}$$

$\Rightarrow$ “Fluidified” solution in $C_{\text{max}}$?
Equations (1/2)

- Initialization (packets leave node $k$):
  $$\sum_{j|(k,j)\in A} n_{k,j}^l = n_{k,l}^k$$

- Reception (packets reach node $l$):
  $$\sum_{i|(i,l)\in A} n_{i,l}^k = n_{k,l}^k$$

- Conservation law (crossing intermediate node $i$):
  $$\sum_{i|(i,j)\in A} n_{i,j}^k = \sum_{i|(j,i)\in A} n_{j,i}^k \quad \forall (k, l), j \neq k, j \neq l$$
Initialization (packets leave node $k$):
$$\sum_{j| (k,j) \in A} n_{k,j}^{k,l} = n_{k,l}^{k,l}$$

Reception (packets reach node $l$):
$$\sum_{i| (i,l) \in A} n_{i,l}^{k,l} = n_{k,l}^{k,l}$$

Conservation law (crossing intermediate node $i$):
$$\sum_{i| (i,j) \in A} n_{i,j}^{k,l} = \sum_{i| (j,i) \in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$
Equations (1/2)

- **Initialization (packets leave node $k$):**
  \[ \sum_{j|(k,j)\in A} n_{k,j} = n^{k,l} \]

- **Reception (packets reach node $l$):**
  \[ \sum_{i|(i,l)\in A} n_{i,l} = n^{k,l} \]

- **Conservation law (crossing intermediate node $i$):**
  \[ \sum_{i|(i,j)\in A} n_{i,j} = \sum_{i|(j,i)\in A} n_{j,i} \quad \forall (k, l), j \neq k, j \neq l \]
Equations (2/2)

- Congestion
  
  \[ C_{i,j} = \sum_{(k,l)|n^k,l > 0} n_i^{k,l} \]

- Objective function

  \[ C_{\text{max}} \geq C_{i,j}, \quad \forall i,j \]

  Minimize \( C_{\text{max}} \)

Linear program in rational numbers: polynomial-time solution. In practice use GLPK, Maple, Mupad ...
Congestion

\[ C_{i,j} = \sum_{(k,l) | n^k,l > 0} n_{i,j}^{k,l} \]

Objective function

\[ C_{\text{max}} \geq C_{i,j}, \quad \forall i, j \]

Minimize \( C_{\text{max}} \)

**Linear program in rational numbers:** polynomial-time solution. In practice use GLPK, Maple, Mupad . . .
Routing algorithm

- Compute optimal solution $C_{\text{max}}$, $n_{i,j}^{k,l}$ of previous linear program
- Periodic schedule:
  - Define $\Omega = \sqrt{C_{\text{max}}}$
  - Use $\left\lceil \frac{C_{\text{max}}}{\Omega} \right\rceil$ periods of length $\Omega$
  - During each period, edge $(i, j)$ forwards (at most)
    $$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l} \Omega}{C_{\text{max}}} \right\rfloor$$
    packets that go from $k$ to $l$
- Clean-up: sequentially process residual packets inside network
Performance

- Schedule is feasible
- Schedule is asymptotically optimal:

\[ C_{\text{max}} \leq C^* \leq C_{\text{max}} + O(\sqrt{C_{\text{max}}}) \]
Why does it work?

- Relaxation of objective function
- Rational number of packets in LP formulation
- Periods long enough so that rounding down to integer numbers has negligible impact
- Periods numerous enough so that loss in first and last periods has negligible impact
- Periodic schedule, described in compact form
Outline

1 Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2 Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3 Pipeline workflows

4 Models and real life

5 Conclusion
Master-worker tasking: framework

Heterogeneous resources
- Processors of different speeds
- Communication links with various bandwidths

Large number of independent tasks to process
- Tasks are atomic
- Tasks have same size

Single data repository
- One master initially holds data for all tasks
- Several workers arranged along a star, a tree or a general graph
Application examples

- Monte Carlo methods
- SETI@home
- Factoring large numbers
- Searching for Mersenne primes
- Particle detection at CERN (LHC@home)
- ... and many others: see BOINC at http://boinc.berkeley.edu
Makespan vs. steady state

Two different problems

- **Makespan**: Maximize total number of tasks processed within a time-bound
- **Steady state**: Determine *periodic task allocation* which maximizes total throughput
Example

Data starts here

My computer

Internet Gateway

Partner site

Cluster Host

Intermediate nodes can compute too

Participating PC's and workstations
Example

A is the root of the tree; all tasks start at A

Time for sending one task from A to B

Time for computing one task in C
Example
Example

A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute

Time →

C receive
C compute
C send

D receive
D compute

A compute
A send
B receive
B compute
C receive
C compute
C send

D receive
D compute

A 3

B 2

C 6

D 2

1

2

3

Yves.Robert@ens-lyon.fr February 8, 2008
Example
Example
Example

Steady-state: 7 tasks every 6 time units

Startup

Repeated pattern

Clean-up

A compute
A send

B receive
B compute

C receive
C compute
C send

D receive
D compute

Steady state: 7 tasks every 6 time units
Rule of the game

- Master sends tasks to workers **sequentially**, and without preemption
- Full computation/communication overlap for each worker
- Worker $P_i$ receives a task in $c_i$ time-units
- Worker $P_i$ processes a task in $w_i$ time-units
Equations

- Worker $P_i$ executes $\alpha_i$ tasks per time-unit
- Computations: $\alpha_i w_i \leq 1$
- Communications: $\sum_i \alpha_i c_i \leq 1$
- Objective: maximize throughput

$$\rho = \sum_i \alpha_i$$
Solution

- Faster-communicating workers first: $c_1 \leq c_2 \leq \ldots$
- Make full use of first $q$ workers, where $q$ largest index s.t.
  \[
  \sum_{i=1}^{q} \frac{c_i}{w_i} \leq 1
  \]
- Make partial use of next worker $P_{q+1}$
- **Discard** other workers

**Bandwidth-centric strategy**
- Delegate work to the fastest communicating workers
- It doesn’t matter if these workers are computing slowly
- Slow workers will not contribute much to overall throughput
Example

Tasks
6 tasks to $P_1$
3 tasks to $P_2$
2 tasks to $P_3$

Communication
$6c_1 = 6$
$3c_2 = 6$
$2c_3 = 6$

Computation
$6w_1 = 18$
$3w_2 = 18$
$2w_3 = 2$

11 tasks every 18 time-units ($\rho = 11/18 \approx 0.6$)
Example

**Fully active**

Discarded

😊 Compare to purely greedy (demand-driven) strategy!
5 tasks every 36 time-units ($\rho = 5/36 \approx 0.14$)

Even if resources are cheap and abundant, resource selection is key to performance.
Extension to trees

Fully used node
Partially used node
Idle node

Resource selection based on local information (children)
Does this really work?

- Can we deal with arbitrary platforms (including cycles)?
- Can we deal with return messages?
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?
Does this really work?

- Can we deal with arbitrary platforms (including cycles)? **Yes**
- Can we deal with return messages?
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?
Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages?
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?
Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages? Yes
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?
Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages? Yes
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?
Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages? Yes
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)? Yes, I mean, almost!
LP formulation still works well ... 

Conservation law

$$\forall m, n \sum_j \text{sent}(P_j \rightarrow P_i, e_{mn}) + \text{executed}(P_i, T_m) = \text{executed}(P_i, T_n) + \sum_k \text{sent}(P_i \rightarrow P_k, e_{mn})$$

Computations

$$\sum_m \text{executed}(P_i, T_m) \times \text{flops}(T_m) \times w_i \leq 1$$

Outgoing communications

$$\sum_m \sum_n \sum_j \text{sent}(P_j \rightarrow P_i, e_{mn}) \times \text{bytes}(e_{mn}) \times c_{ij} \leq 1$$
...but schedule reconstruction is harder

- Actual (cyclic) schedule obtained in polynomial time
- Asymptotic optimality
- A couple of practical problems (large period, # buffers)
- No local scheduling policy
The beauty of steady-state scheduling

**Rationale** Maximize throughput (total load executed per period)

**Simplicity** Relaxation of makespan minimization problem
- Ignore initialization and clean-up phases
- Precise ordering of tasks/messages not needed
- Characterize resource activity per time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving from or sending to which neighbor?

**Efficiency** Optimal throughput $\Rightarrow$ optimal schedule (up to a constant number of tasks)

Periodic schedule, described in compact form
$\Rightarrow$ compiling a loop instead of a DAG!
Lesson learnt?

Resource selection is mandatory

...implementation may still be dynamic, provided that static allocation is enforced by scheduler

Example: demand-driven assignment of enrolled workers
Lesson learnt?

Resource selection is mandatory

...implementation may still be dynamic, provided that static allocation is enforced by scheduler

Example: demand-driven assignment of enrolled workers
Outline

1 Parallel algorithms
   • Independent tasks
   • A simple tiling problem
   • Matrix product (ScaLAPACK)
   • Matrix product (master-slave)
   • Iterative algorithms

2 Scheduling
   • Background: scheduling DAGs
   • Packet routing
   • Steady-state scheduling
   • Multiple applications

3 Pipeline workflows

4 Models and real life

5 Conclusion
Scheduling multiple applications

- Large-scale platforms not likely to be exploited in dedicated mode/single application
- Investigate scenarios in which multiple applications are simultaneously executed on the platform
  ⇒ competition for CPU and network resources
Target problem

- Large complex platform: several clusters and backbone links
- One (divisible load) application running on each cluster
- Which fraction of the job to delegate to other clusters?
- Applications have different communication-to-computation ratios
- How to ensure fair scheduling and good resource utilization?
Linear program

\[
\text{MINIMIZE } \min_k \left\{ \frac{\alpha_k}{\pi_k} \right\},
\]

\text{UNDER THE CONSTRAINTS}

\[
\begin{align*}
(1a) \quad \forall C^k, & \quad \sum_l \alpha_{k,l} = \alpha_k \\
(1b) \quad \forall C^k, & \quad \sum_l \alpha_{l,k} \cdot \tau_l \leq s_k \\
(1c) \quad \forall C^k, & \quad \sum_{l \neq k} \alpha_{k,l} \cdot \delta_k + \sum_{j \neq k} \alpha_{j,k} \cdot \delta_j \leq g_k \\
(1d) \quad \forall l_i, & \quad \sum_{l_i \in L_{k,l}} \beta_{k,l} \leq \text{max-connect}(l_i) \\
(1e) \quad \forall k, l, & \quad \alpha_{k,l} \cdot \delta_k \leq \beta_{k,l} \times g_{k,l} \\
(1f) \quad \forall k, l, & \quad \alpha_{k,l} \geq 0 \\
(1g) \quad \forall k, l, & \quad \beta_{k,l} \in \mathbb{N}
\end{align*}
\]
Approach

- Solution to *rational* linear problem as comparator/upper bound
- Several heuristics, greedy and LP-based
- Use Tiers as topology generator, and then $SIMGRID$
Methodology (cont’d)

| $K$ | $5, 7, \ldots, 90$ |
| $\log(bw(l_k)), \log(g_k)$ | normal ($mean=\log(2000), std=\log(10)$) |
| $s_k$ | uniform, 1000 — 10000 |
| max-connect, $\delta_k, \tau_k, \pi_k$ | uniform, 1 — 10 |

Platform parameters used in simulation
Hints for implementation

- Participants sharing resources in a Virtual Organization
- Centralized broker managing applications and resources
- Broker gathers all parameters of LP program
- Priority factors
- Various policies and refinements possible
  ⇒ e.g. fixed number of connections per application
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion

Yves.Robert@ens-lyon.fr  February 8, 2008
The application

- Consecutive data-sets fed into pipeline
- **Period** $T_{\text{period}} = \text{time interval between beginning of execution of two consecutive data sets}$
- **Latency** $T_{\text{latency}} = \text{time elapsed between beginning and end of execution for a given data set}$
Open problems

**Single workflow**
- Period/latency bi-criteria optimization
- Robust mappings
- Data-parallel stages (decreases latency)
- Replicated stages (decreases period & increases robustness)

**Several (concurrent) workflows**
- Competition for CPU and network resources
- Fairness between applications (max-min throughput, max stretch)
- Sensitivity to application/platform parameter changes
Open problems

Single workflow

- Period/latency bi-criteria optimization
- Robust mappings
- Data-parallel stages (decreases latency)
- Replicated stages (decreases period & increases robustness)

Several (concurrent) workflows

- Competition for CPU and network resources
- Fairness between applications (max-min throughput, max stretch)
- Sensitivity to application/platform parameter changes
Lesson learnt?

Period, latency, stretch, robustness, fairness and combination lead to difficult optimization problems.

Lot of work for young and talented algorithmicians 😊

Example: almost everything yet to be done!
Lesson learnt?

Period, latency, stretch, robustness, fairness and combination lead to difficult optimization problems

Lot of work for young and talented algorithmicians 😊

Example: almost everything yet to be done!
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion
For regular problems, the structure of the task graph (nodes and edges) only depends upon the application, not upon the target platform.

Problems arise from weights, i.e. the estimation of execution and communication times.

Classical answer: “use the past to predict the future”

Divide scheduling into phases, during which machine and network parameters are collected (with NWS).

⇒ This information guides scheduling decisions for next phase.
Experiments versus simulations

- Real experiments difficult to drive (genuine instability of non-dedicated platforms)
- Simulations ensure reproducibility of measured data
- Key issue: run simulations against a realistic environment
- *Trace-based simulation*: record platform parameters today, and simulate the algorithms tomorrow, against recorded data
- Use **SIMGRID**, an event-driven simulation toolkit
SimGrid traces

See http://simgrid.gforge.inria.fr/
Across physical links

Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Across physical links

Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Across physical links

Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Bar-Noy, Guha, Naor, Schieber:
occupation time of sender $P_u$ independent of target $P_v$

not fully multi-port model, but allows for starting a new transfer
from $P_u$ without waiting for previous one to finish
Bhat, Raghavendra and Prasanna:
same parameters for sender $P_u$, link $e_{u,v}$ and receiver $P_v$

Two flavors:
- bidirectional: simultaneous send and receive transfers allowed
- unidirectional: only one send or receive transfer at a given time-step
How to model a file transfer along a path?
How to model a file transfer along a path?

\[ I_1 \]
\[ I_2 \]
\[ I_3 \]
How to model a file transfer along a path?

Store & Forward : bad model for contention
How to model a file transfer along a path?
How to model a file transfer along a path?

WormHole: computation intensive (packets), not that realistic
How to model a file transfer along a path?

\[ \forall l \in L, \sum_{r \in R \text{ s.t. } l \in r} \rho_r \leq c_l \]

Analytical model
How to model a file transfer along a path?

∀l ∈ L, \[ \sum_{r \in R \text{ s.t. } l \in r} \rho_r \leq c_l \]

Max-Min Fairness \[ \text{maximize } \min_{r \in R} \rho_r \]

Proportional Fairness \[ \text{maximize } \sum_{r \in R} \rho_r \log(\rho_r) \]

MCT minimization \[ \text{maximize } \min_{r \in R} \frac{1}{\rho_r} \]

TCP behavior Close to max-min.

In SIMGRID: max-min + bound by \(1/RTT\)
Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open \( i \) TCP connections, receive \( bw(i) \) bandwidth per connection
- \( bw(i) = bw(1)/i \) on a LAN
- Experimental evidence \( \rightarrow bw(i) = bw(1) \) on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: \( bw(i) = \frac{bw(1)}{1 + (i - 1) \cdot \gamma} \)
  - \( \gamma = 1 \) for a perfect LAN, \( \gamma = 0 \) for a perfect WAN
Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open $i$ TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible

- Better model: $bw(i) = \frac{bw(1)}{1 + (i - 1)\cdot\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN
Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open $i$ TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow$ $bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \frac{bw(1)}{1 + (i - 1)\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN
Sample large-scale platform

Accounts for Hierarchy + BW sharing
Assumes knowledge of Routing + Backbone bw + CPU speed
A first trial

Clusters and backbone links
Clusters

- $K$ clusters $C^k$, $1 \leq k \leq K$
- $C^k_{\text{master}}$ front-end processor
- $C^k_{\text{router}}$ router to external world
- $s_k$ cumulated speed of $C^k$
- $g_k$ bandwidth of the LAN link ($\gamma = 1$) from $C^k_{\text{master}}$ to $C^k_{\text{router}}$
Network

- Set $\mathcal{R}$ of routers and $\mathcal{B}$ of backbone links $l_i$
- $bw(l_i)$ bandwidth available for a new connection
- $\text{max-connect}(l_i)$ max. number of connections that can be opened
- Fixed routing: path $L_{k,i}$ of backbones from $C^k_{\text{router}}$ to $C^l_{\text{router}}$
How to cope with uncertainties and dynamicity? (1)

Sensibility analysis

- Asses the impact of uncertainties on existing solutions

Design robust solutions

- Robust optimization
  A robust solution remains “close” to optimal for all scenarios

- Internet-based computing
  - No knowledge on task execution times
  - Minimize risk taken while making any scheduling decision
How to cope with uncertainties and dynamicity? (2)

Stochastic models

1. What are the relevant stochastic models?
   Most characteristics remain to be studied and modeled

2. How can we use them?
   Chance-constrained programming?
   Other mathematical tools?
Outline

1. Parallel algorithms
   - Independent tasks
   - A simple tiling problem
   - Matrix product (ScaLAPACK)
   - Matrix product (master-slave)
   - Iterative algorithms

2. Scheduling
   - Background: scheduling DAGs
   - Packet routing
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Models and real life

5. Conclusion
Tools for the road

- Forget absolute makespan minimization
- Resource selection mandatory
- Divisible load (fractional tasks)
- Single application: period / latency / power / robustness
- Several applications: max-min fairness, MAX stretch
- Linear programming: absolute bound to assess heuristics
Scheduling for large-scale platforms

- If platform is well identified and relatively stable, try to:
  (i) accurately model hierarchical structure
  (ii) design well-suited and robust scheduling algorithms

- If platform is not stable enough, or if it evolves too fast, dynamic schedulers are the only option

- Otherwise, grab any opportunity to

  ★★★ inject static knowledge into dynamic schedulers ★★★

- Is this opportunity a niche?
- Does it encompass a wide range of applications?
Who cares about algorithm design and scheduling?

Heard it through the grapevine?

Scheduling is “this thing that people in academia like to think about but that people who do real stuff sort of ignore”
Who cares about algorithm design and scheduling?

Heard it through the grapevine?

Scheduling is “this thing that people in academia like to think about but that people who do real stuff sort of ignore”

Let’s prove this wrong?!