Algorithms and scheduling techniques for heterogeneous platforms

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,
Jeanne Ferrante, Arnaud Legrand, Loris Marchal, Frédéric Vivien

Austin, September 2007
Community of users
e.g. colleagues from various universities around the world

They share disk space:
- publications and research reports on the web
- CVS repositories for distributed collaborative work
- information is ubiquitous

They don’t share CPU resources:
- each user in the community has access to many machines
- difficult to “assemble” large-scale distributed platforms
- when setup, difficult to use
Community of users
e.g. colleagues from various universities around the world

They share disk space:
- publications and research reports on the web
- CVS repositories for distributed collaborative work
- information is ubiquitous

They don’t share CPU resources:
- each user in the community has access to many machines
- difficult to “assemble” large-scale distributed platforms
- when setup, difficult to use
Community of users
e.g. colleagues from various universities around the world

They share disk space:
- publications and research reports on the web
- CVS repositories for distributed collaborative work
- information is ubiquitous

They don’t share CPU resources:
- each user in the community has access to many machines
- difficult to “assemble” large-scale distributed platforms
- when setup, difficult to use
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?

Heterogeneity everywhere:
Processor speeds and memory capacities
Communication startups and link bandwidths
Large-scale architectures

- Gateways
- Grid portals
- Peer-to-peer networks
- Volunteer computing platforms
- Virtual organizations
- ...

Software problems
Large-scale architectures

- Gateways
- Grid portals
- Peer-to-peer networks
- Volunteer computing platforms
- Virtual organizations
- ...

Software problems

- System images
- Middleware deployment
- Administration issues
- Security concerns
- Platform discovery
- Platform parameters
Large-scale architectures

Software problems

- Gateways
- Grid portals
- Peer-to-peer networks
- Volunteer computing platforms
- Virtual organizations
- ...

Sorry!

Not today’s talk?!
Heterogeneity everywhere

Algorithmic and scheduling issues

- Processor speed
- Memory capacity
- Communication startup
- Link bandwidth

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

On heterogeneous platforms, it gets worse.
Algorithmic and scheduling issues

- Processor speed
- Memory capacity
- Communication startup
- Link bandwidth

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

On heterogeneous platforms, it gets worse.
Heterogeneity everywhere

**Algorithmic and scheduling issues**

- Processor speed
- Memory capacity
- Communication startup
- Link bandwidth

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

On heterogeneous platforms, it gets worse
Heterogeneity everywhere

Algorithmic and scheduling issues

- Processor speed
- Memory capacity
- Communication startup
- Link bandwidth

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

On heterogeneous platforms, it gets worse
Outline

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

2. Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Limitations

5. Conclusion
Outline

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

2. Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Limitations

5. Conclusion
Outline

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

2 Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3 Pipeline workflows

4 Limitations

5 Conclusion
Independent chunks

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

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

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

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

\[
\begin{array}{ccccccc}
\text{Task} & n_1 & n_2 & n_3 & T_{\text{exe}} & \text{Selected proc.} \\
0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 3 & 2 \\
2 & 1 & 1 & 0 & 5 & 1 \\
3 & 2 & 1 & 0 & 6 & 3 \\
4 & 2 & 1 & 1 & 8 & 1 \\
5 & 3 & 1 & 1 & 9 & 2 \\
6 & 3 & 2 & 1 & 10 & 1 \\
7 & 4 & 2 & 1 & 12 & 1 \\
8 & 5 & 2 & 1 & 15 & 2 \\
9 & 5 & 3 & 1 & 15 & 3 \\
10 & 5 & 3 & 2 & 16 & \end{array}
\]
Dynamic programming

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

Task | $n_1$ | $n_2$ | $n_3$ | $T_{exe}$ | Selected proc.
--- | --- | --- | --- | --- | ---
0 | 0 | 0 | 0 | | 1 |
1 | 1 | 0 | 0 | 3 | 2 |
2 | 1 | 1 | 0 | 5 | 1 |
3 | 2 | 1 | 0 | 6 | 3 |
4 | 2 | 1 | 1 | 8 | 1 |
5 | 3 | 1 | 1 | 9 | 2 |
6 | 3 | 2 | 1 | 10 | 1 |
7 | 4 | 2 | 1 | 12 | 1 |
8 | 5 | 2 | 1 | 15 | 2 |
9 | 5 | 3 | 1 | 15 | 3 |
10 | 5 | 3 | 2 | 16 |
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: $w_1 = 3$, $w_2 = 5$, 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:** $w_1 = 3$, $w_2 = 5$, 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>$T_{exe}$</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>$T_{exe}$</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: \( w_1 = 3, w_2 = 5, \) and \( w_3 = 8 \)

\[
\begin{array}{c}
P_1 \bullet \bullet \bullet \circ \circ \circ \bullet \bullet \bullet \circ \circ \circ \bullet \bullet \bullet \circ \circ \circ \\
P_2 \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \blacktriangle \\
P_3 \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \\
\end{array}
\]

<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  Heteropar'07  Algorithms and scheduling techniques
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_{\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)

2 Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3 Pipeline workflows

4 Limitations

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

\[
\begin{align*}
6 & \rightarrow 6 \\
5 & \rightarrow 6 \\
4 & \rightarrow 5 \rightarrow 6 \\
3 & \rightarrow 4 \rightarrow 5 \\
2 & \rightarrow 3 \rightarrow 4 \\
1 & \rightarrow 2 \rightarrow 3 \\
\end{align*}
\]
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

\[ \ldots \]

\[
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5 \\
6 \end{array}
\rightarrow
\begin{array}{c}
2 \\
3 \\
4 \\
5 \\
6 \\
1 \end{array}
\]

Stepwise execution
Allocation strategy (1/3)

Use column-wise allocation to enhance locality

Stepwise execution
Use column-wise allocation to enhance locality

\[ \ldots \]

\[ \begin{array}{c}
   6 \\
   5 \rightarrow 6 \\
   4 \rightarrow 5 \rightarrow 6 \\
   3 \rightarrow 4 \rightarrow 5 \\
   2 \rightarrow 3 \rightarrow 4 \\
   1 \rightarrow 2 \rightarrow 3 \\
\end{array} \ldots \]

Stepwise execution
Allocation strategy (2/3)

With column-wise allocation,

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

- Greedy (demand-driven) allocation \( \Rightarrow \) 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 \( \Rightarrow \text{slowdown} \) !

• Execution progresses at the pace of the slowest processor 😞
With 3 processors, $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$:

$$T_{exe} \approx \frac{8}{3} N_1 N_2 \approx 2.67 N_1 N_2$$

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

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

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)

2 Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3 Pipeline workflows

4 Limitations

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
ScaLAPACK algorithm on (homogeneous) 2D grids (1/2)

- $C = AB$ on a $p \times q$ processor grid
- Granularity: one element = one square $r \times r$ block
- Each matrix is partitioned into $p \times q$ 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
**Objective:** \[ \max r_i \times t_{ij} \times c_j \leq 1 \left( \sum_{i=1}^{p} r_i \right) \times \left( \sum_{j=1}^{q} c_j \right) \]

Maximize total number of elements processed within one time unit
2D load balancing (2/2)

Given \( n = p \times q \) processors, how to arrange them along a 2D grid of size \( p \times q \) ...

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

- Search among all possible arrangements of the \( p \times q \) processors as a \( p \times q \) grid
- For each arrangement, solve optimization problem
- **NP-hard 😞**
Matrix product on heterogeneous clusters

Matrix product with 13 heterogeneous processors
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)

2. Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Limitations

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**
Application model

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

- *Star network* master $M$ and $p$ workers $P_i$
- $X \cdot w_i$ time-units for $P_i$ to execute a task of size $X$
- $X \cdot c_i$ 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 $m = 21$
Example with $m = 21$
Example with $m = 21$

|   | $A_{11}$ | $B_{11}$ | $B_{12}$ | $B_{13}$ | $B_{14}$ | $C_{11}$ | $C_{12}$ | $C_{13}$ | $C_{14}$ | $C_{21}$ | $C_{22}$ | $C_{23}$ | $C_{24}$ | $C_{31}$ | $C_{32}$ | $C_{33}$ | $C_{34}$ | $C_{41}$ | $C_{42}$ | $C_{43}$ | $C_{44}$ |
|---|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|   |          |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |

Yves.Robert@ens-lyon.fr  
Heteropar'07  
Algorithms and scheduling techniques  
34/82
Example with $m = 21$

\[
\begin{array}{|c|c|c|c|}
\hline
B_{11} & B_{12} & B_{13} & B_{14} \\
\hline
\end{array}
\]

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

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

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

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

\( c = 2, \, w = 4.5, \, \mu = 4, \, t = 100, \) enroll \( \mathcal{P} = 5 \) workers

\[
\mathcal{P} \times \mu^2 C \quad \mathcal{P} \times \mu(A, B) \mathcal{P} \times \mu(A, B) \quad \mathcal{P} \times \mu(A, B) \quad \mathcal{P} \times \mu^2 C
\]
Performance

- Communication-to-computation ratio:

\[
\frac{2}{t} + \frac{2}{\mu} \rightarrow \frac{2}{\sqrt{m}}
\]

- Close to lower bound

- Enroll \( \Psi \leq p \) workers, where

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

In the example, \( \Psi = \lceil 4.5 \rceil \)

- Typically, \( c = q^2 \tau_c \) and \( w = q^3 \tau_a \)
  → resource selection \( \Psi = \left\lfloor \mu q \frac{\tau_a}{2 \tau_c} \right\rfloor \)
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)
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)
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)

2 Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3 Pipeline workflows

4 Limitations

5 Conclusion
Outline

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

2. Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Limitations

5. Conclusion
Scheduling on 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 $T_j$ 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 $e_{ij} : T_i \rightarrow T_j$
Dependence 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 list scheduling 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
  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)

2 Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3 Pipeline workflows

4 Limitations

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

Yves.Robert@ens-lyon.fr Heteropar'07

Algorithms and scheduling techniques 49/82
Example

Time for sending one task from A to B

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

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→

A 3
B 2
C 6
D 2
Example

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

Time →

1 2 3
Example
Example

A compute
A send

B receive
B compute

C receive
C compute
C send

D receive
D compute

Time→
Example

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
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

Introduction

Parallel algorithms

Scheduling

Pipeline workflows

Limitations

Conclusion

Example

Fully active

Discarded

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Communication</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 tasks to $P_1$</td>
<td>$6c_1 = 6$</td>
<td>$6w_1 = 18$</td>
</tr>
<tr>
<td>3 tasks to $P_2$</td>
<td>$3c_2 = 6$</td>
<td>$3w_2 = 18$</td>
</tr>
<tr>
<td>2 tasks to $P_3$</td>
<td>$2c_3 = 6$</td>
<td>$2w_3 = 2$</td>
</tr>
</tbody>
</table>

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 \( \left( \rho = \frac{5}{36} \approx 0.14 \right) \)

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

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

Computations

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

Outgoing communications

\[ \sum_{m,n} \sum_j \text{sent}(P_j \rightarrow P_i, e_{mn}) \times \text{bytes}(e_{mn}) \times c_{ij} \leq 1 \]
... 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 ⇒ optimal schedule (up to a constant number of tasks)

Periodic schedule, described in compact form
⇒ 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)

2. Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
     - Multiple applications

3. Pipeline workflows

4. Limitations

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
  \[ \Rightarrow \text{competition} \] 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

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

\[\text{UNDER THE CONSTRAINTS}\]

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

Several heuristics, greedy and LP-based

Use Tiers as topology generator, and then $S_{IMGIRID}$
Methodology (cont’d)

Platform parameters used in simulation

<table>
<thead>
<tr>
<th>$K$</th>
<th>$\log(bw(l_k)), \log(g_k)$</th>
<th>$s_k$</th>
<th>max-connect, $\delta_k, \tau_k, \pi_k$</th>
</tr>
</thead>
</table>

- $K$: 5, 7, ..., 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
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
  \[\Rightarrow\] 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)

2. Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3. Pipeline workflows

4. Limitations

5. Conclusion
Consecutive data-sets fed into pipeline

**Period** $T_{\text{period}}$ = time interval between beginning of execution of two consecutive data sets

**Latency** $T_{\text{latency}}$ = time elapsed between beginning and end of execution for a given data set
Rule of the game

- Platform = fully interconnected set of different-speed processors
- Map each pipeline stage on a single processor
- **Objective:** minimize period and/or latency

Several mapping strategies

\[
S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_k \rightarrow \ldots \rightarrow S_n
\]

Pipeline application
Rule of the game

- Platform = fully interconnected set of different-speed processors
- Map each pipeline stage on a single processor
- Objective: minimize period and/or latency
- Several mapping strategies

Pipeline application
Rule of the game

- Platform = fully interconnected set of different-speed processors
- Map each pipeline stage on a single processor
- **Objective:** minimize period and/or latency
- Several mapping strategies

**One-to-one Mapping**
Rule of the game

- Platform = fully interconnected set of different-speed processors
- Map each pipeline stage on a single processor
- **Objective**: minimize period and/or latency
- **Several mapping strategies**

![Interval Mapping Diagram](image-url)
Rule of the game

- Platform = fully interconnected set of different-speed processors
- Map each pipeline stage on a single processor
- Objective: minimize period and/or latency
- Several mapping strategies

General Mapping
Chains-on-chains

Load-balance **contiguous** tasks

5 7 3 4 8 1 3 8 2 9 7 3 5 2 3 6
Chains-on-chains

Load-balance contiguous tasks

5 7 3 4 8 1 3 8 2 9 7 3 5 2 3 6

With $p = 4$ identical processors?
Chains-on-chains

Load-balance **contiguous** tasks

5 7 3 4 8 1 3 8 2 9 7 3 5 2 3 6

With $p = 4$ identical processors?

5 7 3 4 | 8 1 3 8 | 2 9 7 | 3 5 2 3 6

$T_{\text{period}} = 20$
Chains-on-chains

Load-balance **contiguous** tasks

5 7 3 4 8 1 3 8 2 9 7 3 5 2 3 6

With $p = 4$ identical processors?

5 7 3 4 | 8 1 3 8 | 2 9 7 | 3 5 2 3 6

$T_{\text{period}} = 20$

NP-hard for different-speed processors, even without communications
Some combinatorics for the road

\[
S_1 \to S_2 \to S_3 \to S_4
\]

14 \to 4 \to 2 \to 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?
\[ T_{\text{period}} = 7, \ S_1 \to P_1, \ S_2S_3 \to P_2, \ S_4 \to P_3 \ (T_{\text{latency}} = 17) \]

Optimal latency?
\[ T_{\text{latency}} = 12, \ S_1S_2S_3S_4 \to P_1 \ (T_{\text{period}} = 12) \]

Min. latency if \( T_{\text{period}} \leq 10? \)
\[ T_{\text{latency}} = 14, \ S_1S_2S_3 \to P_1, \ S_4 \to P_2 \]
Some combinatorics for the road

\[
S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \\
14 \quad 4 \quad 2 \quad 4
\]

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?
\( T_{\text{period}} = 7, \ S_1 \rightarrow P_1, \ S_2S_3 \rightarrow P_2, \ S_4 \rightarrow P_3 \ (T_{\text{latency}} = 17) \)

Optimal latency?
\( T_{\text{latency}} = 12, \ S_1S_2S_3S_4 \rightarrow P_1 \ (T_{\text{period}} = 12) \)

Min. latency if \( T_{\text{period}} \leq 10? \)
\( T_{\text{latency}} = 14, \ S_1S_2S_3 \rightarrow P_1, \ S_4 \rightarrow P_2 \)
Some combinatorics for the road

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]

\[ 14 \quad 4 \quad 2 \quad 4 \]

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?

\[ T_{\text{period}} = 7, \ S_1 \rightarrow P_1, \ S_2S_3 \rightarrow P_2, \ S_4 \rightarrow P_3 \ (T_{\text{latency}} = 17) \]

Optimal latency?

\[ T_{\text{latency}} = 12, \ S_1S_2S_3S_4 \rightarrow P_1 \ (T_{\text{period}} = 12) \]

Min. latency if \( T_{\text{period}} \leq 10? \)

\[ T_{\text{latency}} = 14, \ S_1S_2S_3 \rightarrow P_1, \ S_4 \rightarrow P_2 \]
Some combinatorics for the road

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]
\[ 14 \quad 4 \quad 2 \quad 4 \]

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

**Optimal period?**
\[ T_{\text{period}} = 7, \ S_1 \rightarrow P_1, \ S_2S_3 \rightarrow P_2, \ S_4 \rightarrow P_3 \ (T_{\text{latency}} = 17) \]

**Optimal latency?**
\[ T_{\text{latency}} = 12, \ S_1S_2S_3S_4 \rightarrow P_1 \ (T_{\text{period}} = 12) \]

**Min. latency if \( T_{\text{period}} \leq 10?**
\[ T_{\text{latency}} = 14, \ S_1S_2S_3 \rightarrow P_1, \ S_4 \rightarrow P_2 \]
Some combinatorics for the road (cont’d)

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]

14 4 2 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

**Replicate** interval \([S_u...S_v]\) on \( P_1, \ldots, P_q \)

\[ T_{\text{period}} = \frac{\sum_{k=u}^{v} w_k}{q \times \min_i(s_i)} \quad \text{and} \quad T_{\text{latency}} = q \times T_{\text{period}} \]
Some combinatorics for the road (cont’d)

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]

14 4 2 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Data Parallelize single stage \( S_k \) on \( P_1, \ldots, P_q \)

\[ S \ (w = 16) \]

\[ \begin{array}{c}
\bullet\bullet\bullet \\
\bullet\bullet\bullet
\end{array} \]

\[ \Rightarrow \]

\[ \begin{array}{c}
P_1 \ (s_1 = 2) : \bullet\bullet\bullet\bullet\bullet\bullet
\\
P_2 \ (s_2 = 1) : \bullet\bullet\bullet
\\
P_3 \ (s_3 = 1) : \bullet\bullet\bullet\bullet
\end{array} \]

\[ T_{\text{period}} = \frac{w_k}{\sum_{i=1}^{q} s_i} \] and \( T_{\text{latency}} = T_{\text{period}} \)
Some combinatorics for the road (cont’d)

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]

14 4 2 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?
Some combinatorics for the road (cont’d)

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]

14 \hspace{1cm} 4 \hspace{1cm} 2 \hspace{1cm} 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?

\[ S_1 \overset{\text{DP}}{\rightarrow} P_1P_2, \ S_2S_3S_4 \overset{\text{REP}}{\rightarrow} P_3P_4 \]

\[ T_{\text{period}} = \max \left( \frac{14}{2+1}, \frac{4+2+4}{2\times1} \right) = 5, \ T_{\text{latency}} = 14.67 \]
Some combinatorics for the road (cont’d)

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]
\[ 14 \quad 4 \quad 2 \quad 4 \]

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?

\[ S_1 \xrightarrow{DP} P_1P_2, \quad S_2S_3S_4 \xrightarrow{REP} P_3P_4 \]

\[ T_{\text{period}} = \max\left( \frac{14}{2+1}, \frac{4+2+4}{2\times1} \right) = 5, \quad T_{\text{latency}} = 14.67 \]

\[ S_1 \xrightarrow{DP} P_2P_3P_4, \quad S_2S_3S_4 \xrightarrow{REP} P_1 \]

\[ T_{\text{period}} = \max\left( \frac{14}{1+1+1}, \frac{4+2+4}{2} \right) = 5, \quad T_{\text{latency}} = 9.67 \ (\text{optimal}) \]
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)

2 Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3 Pipeline workflows

4 Limitations

5 Conclusion
Knowledge of the platform graph

- 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
Sample large-scale platform

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

Clusters and backbone links
A first trial (cont’d)

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
- max-connect$(l_i)$ max. number of connections that can be opened
- Fixed routing: path $L_{k,l}$ of backbones from $C^k_{\text{router}}$ to $C^l_{\text{router}}$
Outline

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

2 Scheduling
   - Scheduling DAGs
   - Steady-state scheduling
   - Multiple applications

3 Pipeline workflows

4 Limitations

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 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?