Think before coding:
static strategies (and dynamic execution)
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, Matthieu Gallet, Arnaud Legrand,
Loris Marchal, Jean-François Pineau, Veronika Rehn, Frédéric Vivien

ICCS 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
Collaborative architectures . . .

- Gateways
- Grid portals
- Peer-to-peer networks
- Volunteer computing platforms
- Virtual organizations
- ...
...all made out from heterogeneous resources

From (good old) parallel architectures...
... all made out from heterogeneous resources

... to heterogeneous clusters ...
... all made out from heterogeneous resources

... and to large-scale grid platforms?
all made out from heterogeneous resources

...and to large-scale grid platforms?

Heterogeneity everywhere:
Processor speeds and memory capacities
Communication startups and link bandwidths
Collaborative architectures (again)

Software problems

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

Yves.Robert@ens-lyon.fr  ICCS 2007
Collaborative architectures (again)

Software problems

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

System images
Middleware deployment
Administration issues
Security concerns
Platform discovery
Platform parameters
Collaborative architectures (again)

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.
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
The dream of the scheduling guy

- Assume the platform is built and ready to use
- Assume exact knowledge of all platform parameters
- Assume exact knowledge of all application parameters

Ready to think before coding! 😊
The dream of the scheduling guy

- Assume the platform is built and ready to use
- Assume exact knowledge of all platform parameters
- Assume exact knowledge of all application parameters

Ready to think before coding! 😊
Outline

1. A simple tiling problem
2. Master-worker
3. Matrix product
4. Pipeline workflows
5. Limitations
6. Conclusion
Outline

1. A simple tiling problem
2. Master-worker
3. Matrix product
4. Pipeline workflows
5. Limitations
6. 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 $1/w_i$

- Assign $n_i$ tasks to $P_i$

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

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

<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:** $w_1 = 3$, $w_2 = 5$, and $w_3 = 8$

![Diagram showing task assignment to processors](image)

<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_{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

ICCS 2007

Static strategies for clusters and grids

11/ 59
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 $w_3 = 8$
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:** \( w_1 = 3, \ w_2 = 5, \text{ and } w_3 = 8 \)

### Task Allocation

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

\[
\begin{align*}
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad
\end{align*}
\]

<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 😞
A simple finite difference problem

- Iteration space: 2D rectangle of size $N_1 \times N_2$
- Dependences between tiles $\left\{ \left( \begin{array}{c} 1 \\ 0 \end{array} \right), \left( \begin{array}{c} 0 \\ 1 \end{array} \right) \right\}$
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
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 (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 \( \Rightarrow \text{slowdown} \) ?!
- Execution progresses at the pace of the slowest processor 😞
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 \(\Rightarrow\) \textbf{slowdown} ?!

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

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

$$T_{\text{opt}} \approx \frac{120}{79} N_1 N_2 \approx 1.52 \frac{N_1 N_2}{N_1 N_2}$$
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 One

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

. . . but of course implementation will be dynamic

Example: demand-driven allocation of blocks of adequate size
Lesson One

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

. . . but of course implementation will be dynamic

Example: demand-driven allocation of blocks of adequate size
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
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

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.

- 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

A compute
A send

B receive
B compute

C receive
C compute
C send

D receive
D compute

Time

1 2 3
Example
Example
Example
Example

A simple tiling problem

Master-worker

Matrix product

Pipeline workflows

Limitations

Conclusion

A compute
A send

B receive
B compute

C receive
C compute
C send

D receive
D compute

Time→

Yves.Robert@ens-lyon.fr ICCS 2007
Example

A simple tiling problem

Master-worker

Matrix product

Pipeline workflows

Limitations

Conclusion

Startup

Repeated pattern

Clean-up

Steady-state: 7 tasks every 6 time units

Yves.Robert@ens-lyon.fr

ICCS 2007

Static strategies for clusters and grids 24/59
**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

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

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

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

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

Computations

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

Outgoing communications

\[ \sum_m \sum_n \sum_j \text{sent}(P_j \to P_i, e_{mn}) \times \text{bytes}(e_{mn}) \times c_{ij} \leq 1 \]
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 Two

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 Two

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. A simple tiling problem
2. Master-worker
3. Matrix product
4. Pipeline workflows
5. Limitations
6. 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
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 $P_i$
\[ \implies \text{at most } m_i \text{ blocks simultaneously stored on worker} \]
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 $P_i$
→ 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 $m = 21$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_{11}$</td>
<td>$C_{12}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example with $m = 21$
Example with \( m = 21 \)

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

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

\[
\begin{array}{cc}
B_{11} & B_{12} \\
\end{array}
\]
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 $m = 21$

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

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

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

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

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

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

c = 2, w = 4.5, \mu = 4, t = 100, enroll \Psi = 5 workers
Performance

- Communication-to-computation ratio:

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

- Close to lower bound

- Enroll \( p \leq \mathcal{P} \) workers, where

\[
\mathcal{P} = \left\lceil \frac{\mu w}{2c} \right\rceil
\]

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

- Typically, \( c = q^2 \tau_c \) and \( w = q^3 \tau_a \)

\( \rightarrow \) resource selection \( \mathcal{P} = \left\lceil \mu q \frac{\tau_a}{2\tau_c} \right\rceil \)
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)
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
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. A simple tiling problem
2. Master-worker
3. Matrix product
4. Pipeline workflows
5. Limitations
6. Conclusion
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** $T_{\text{latency}} = \text{time elapsed between beginning and end of execution for a given data set}$
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
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

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

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**
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**

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

**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

\[
\begin{array}{cccccccccccc}
5 & 7 & 3 & 4 & 8 & 1 & 3 & 8 & 2 & 9 & 7 & 3 & 5 & 2 & 3 & 6 \\
\end{array}
\]

With \( p = 4 \) identical processors?

\[
\begin{array}{cccccccc}
5 & 7 & 3 & 4 & | & 8 & 1 & 3 & 8 & | & 2 & 9 & 7 & | & 3 & 5 & 2 & 3 & 6 \\
\end{array}
\]

\[ T_{\text{period}} = 20 \]
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

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

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

**Optimal period?**
\[ T_{\text{period}} = 7, \; S_1 \rightarrow P_1, \; S_2 S_3 \rightarrow P_2, \; S_4 \rightarrow P_3 \; (T_{\text{latency}} = 17) \]

**Optimal latency?**
\[ T_{\text{latency}} = 12, \; S_1 S_2 S_3 S_4 \rightarrow P_1 \; (T_{\text{period}} = 12) \]

**Min. latency if \( T_{\text{period}} \leq 10?**
\[ T_{\text{latency}} = 14, \; S_1 S_2 S_3 \rightarrow P_1, \; S_4 \rightarrow P_2 \]
Some combinatorics for the road

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

14 4 2 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 \( P_1, \ldots, P_q \)

\[
\begin{align*}
\begin{array}{c}
S_u \ldots S_v \text{ on } P_1: \text{ data sets } 1, 4, 7, \ldots \\
\text{\ldots} S \text{ \ldots} S_u \ldots S_v \text{ on } P_2: \text{ data sets } 2, 5, 8, \ldots \\
\text{\ldots} S \text{ \ldots} S_u \ldots S_v \text{ on } P_3: \text{ data sets } 3, 5, 9, \ldots
\end{array}
\end{align*}
\]

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

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

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

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

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

\[ S \ (w = 16) \quad P_1 \ (s_1 = 2) : \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \]

\[ \Rightarrow \quad P_2 \ (s_2 = 1) : \bullet \bullet \bullet \]

\[ \Rightarrow \quad P_3 \ (s_3 = 1) : \bullet \bullet \bullet \bullet \bullet \]

\[ T_{\text{period}} = \frac{w_k}{\sum_{i=1}^{q} s_i} \quad \text{and} \quad T_{\text{latency}} = T_{\text{period}} \]
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 4 2 4

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

Optimal period?

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

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

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]
\[
\begin{array}{cccc}
14 & 4 & 2 & 4
\end{array}
\]
Interval mapping, 4 processors, \(s_1 = 2\) and \(s_2 = s_3 = s_4 = 1\)

Optimal period?

\[ S_1 \xrightarrow{\text{DP}} P_1P_2, \quad S_2S_3S_4 \xrightarrow{\text{REP}} P_3P_4 \]
\[
T_{\text{period}} = \max\left(\frac{14}{2+1}, \frac{4+2+4}{2\times1}\right) = 5, \quad T_{\text{latency}} = 14.67
\]

\[ S_1 \xrightarrow{\text{DP}} P_2P_3P_4, \quad S_2S_3S_4 \rightarrow P_1 \]
\[
T_{\text{period}} = \max\left(\frac{14}{1+1+1}, \frac{4+2+4}{2}\right) = 5, \quad T_{\text{latency}} = 9.67 \quad \text{(optimal)}
\]
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 (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 (stretch)
- Sensitivity to application/platform parameter changes
Lesson Four

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 Four

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. A simple tiling problem
2. Master-worker
3. Matrix product
4. Pipeline workflows
5. Limitations
6. 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).
  \[\Rightarrow\] This information guides scheduling decisions for next phase.
- Moving from heterogeneous clusters to computational grids causes further problems (even discovering the characteristics of the surrounding computing resources may prove a difficult task).
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 $C^k$, $1 \leq k \leq K$
- $C_{\text{master}}^k$ front-end processor
- $C_{\text{router}}^k$ router to external world
- $s_k$ cumulated speed of $C^k$
- $g_k$ bandwidth of the LAN link ($\gamma = 1$) from $C_{\text{master}}^k$ to $C_{\text{router}}^k$
Network

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