Mapping Heuristics in Heterogeneous Computing

Alexandru Samachisa
Dmitriy Bekker

Multiple Processor Systems (EECC756)

May 18, 2006

Dr. Shaaban
Overview

- Introduction
  - Mapping overview
  - Homogenous computing vs. heterogeneous computing
- Task mapping heuristics (for HC)
  - Overview (static, dynamic, online, batch)
  - DAG used in examples
  - Performance metrics
- Selected algorithms
  - HEFT, CPOP, MCT, Min-min, Max-min, Sufferage
- Conclusion
Mapping Overview

- Mapping (matching and scheduling) of processes onto processors
- Goal
  - Assign process to best suited machine
  - Maximize performance
- NP-complete
- Heuristics exist to optimize mapping performance
- Best heuristic to use depends on
  - Static or dynamic?
  - Task arrival rate
  - Number of tasks
Homogenous vs. Heterogeneous Computing

- **Homogenous computing**
  - Best for fixed task grain size (most common)
  - Support for single mode of parallelism
  - Custom architecture

- **Heterogeneous computing** (HC)
  - Variable task grain size
  - Support for multiple modes of parallelism
  - Machines in cluster may be different

- **Mapping much simpler in homogenous computing**
  - Known task size
  - Known task execution time on any node

- **In HC…**
  - Task size and machine performance varies
  - Mapping heuristic must find optimal (or sub-optimal) machine for each task
Mapping Heuristics

- **Static Mapping**
  - Performed at compile time (off-line)
  - Expected Time to Compute (ETC) is known for all tasks

- **Dynamic Mapping**
  - Performed at run-time
  - ETC only known for currently mapping tasks
  - Two modes
    - On-line (one task mapped at a time)
    - Batch (acquire a number of tasks into meta-task and then map)
## DAG used in examples

![DAG diagram]

### Computation Costs

<table>
<thead>
<tr>
<th>Task</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>13</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>7</td>
<td>16</td>
</tr>
</tbody>
</table>
Performance Metrics

- Expected execution time (EET)
  - Assumes no load on system

- Expected completion time (ECT)
  - $ECT = \text{begin time} + \text{EET}$

- Makespan (maximum ECT)

- Average sharing penalty
  - Based on waiting time of task
  - Good measure of quality of service

- Schedule length ratio (SLR)
  - Makespan normalized to sum of minimum computation costs on minimum critical path (for static mapping)
HEFT and CPOP

- Both static algorithms (know whole DAG)
- Prioritize based on ranks (recursive)
  - Upward rank
    \[
    rank_u(n_i) = \bar{w}_i + \max_{n_j \in \text{succ}(n_i)} (\bar{c}_{i,j} + rank_u(n_j))
    \]
  - Downward rank
    \[
    rank_d(n_i) = \max_{n_j \in \text{pred}(n_i)} \left\{ rank_d(n_j) + \bar{w}_j + \bar{c}_{j,i} \right\}
    \]
**HEFT**

- Goal: schedule task on “best” processor, minimize finish time
- Gives each task in the graph a priority based on the communication costs and computation (decreasing upward rank)
- Iterate through the tasks based on priority and assign the process with the earliest finish time for the task
- Complexity: $O(ep)$
  - $e =$ edges
  - $p =$ processors
- Order $n^2$ complexity for very dense graphs ($n$ is number of tasks)
CPOP

- Similar to HEFT
- Prioritizes the tasks based on communication and computation cost (summation of upward and downward ranks)
- Determines critical path processor (minimized cumulative computation costs of tasks on path)
- Assign critical path task to critical path processor
  - If task not in critical path, assign it to processor which minimizes earliest execution finish time
- Same complexity as HEFT: $O(ep)$
On-line mode

- Each task is assigned as soon as received
- Usually easy to implement since the selection is based on a simple criteria
- Heuristics: Minimum completion time, minimum execution time, switching algorithm, K-percent best, etc.
- The on-line heuristics usually need $O(m)$ time to assign the process
## MCT

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>14</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>N2</td>
<td>40</td>
<td>46</td>
<td>27</td>
</tr>
<tr>
<td>N3</td>
<td>32</td>
<td>34</td>
<td>46</td>
</tr>
<tr>
<td>N4</td>
<td>45</td>
<td>26</td>
<td>44</td>
</tr>
<tr>
<td>N5</td>
<td>44</td>
<td>39</td>
<td>37</td>
</tr>
<tr>
<td>N6</td>
<td>45</td>
<td>42</td>
<td>46</td>
</tr>
<tr>
<td>N7</td>
<td>39</td>
<td>80</td>
<td>61</td>
</tr>
<tr>
<td>N8</td>
<td>62</td>
<td>57</td>
<td>71</td>
</tr>
<tr>
<td>N9</td>
<td>68</td>
<td>69</td>
<td>70</td>
</tr>
<tr>
<td>N10</td>
<td>89</td>
<td>88</td>
<td>97</td>
</tr>
</tbody>
</table>

Assigns each task on arrival to the process that yields the MCT for that task

Complexity: O(m)
Batch mode

- The batch mode heuristics assume that tasks are independent.
- The set of independent tasks that is being processed is called a Meta-task.
- All tasks get remapped at every mapping interval.
- Our assumptions:
  - Each level of dependency is constitutes a meta-task.
  - Once a task is assigned, it will not be remapped at the next mapping event.
Min-min (meta-task 2)

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2</td>
<td>40</td>
<td>46</td>
<td>27</td>
</tr>
<tr>
<td>N3</td>
<td>32</td>
<td>34</td>
<td>28</td>
</tr>
<tr>
<td>N4</td>
<td>31</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>N5</td>
<td>32</td>
<td>33</td>
<td>19</td>
</tr>
<tr>
<td>N6</td>
<td>36</td>
<td>39</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2</td>
<td>40</td>
<td>46</td>
<td>36</td>
</tr>
<tr>
<td>N3</td>
<td>32</td>
<td>34</td>
<td>37</td>
</tr>
<tr>
<td>N5</td>
<td>32</td>
<td>39</td>
<td>28</td>
</tr>
</tbody>
</table>

Complexity: $O(S^2m)$ per meta-task, where $m$ is the number of machines and $S$ is the number of tasks in the meta-task
Complexity: $O(S^2m)$ per meta-task, where $m$ is the number of machines and $S$ is the number of tasks in the meta-task.

### Multiple Processor Systems (EECC756) Spring 2006
Sufferage (meta-task 2)

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2</td>
<td>40</td>
<td>46</td>
<td>27</td>
<td>13</td>
</tr>
<tr>
<td>N3</td>
<td>32</td>
<td>34</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td>N4</td>
<td>31</td>
<td>26</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>N5</td>
<td>32</td>
<td>33</td>
<td>19</td>
<td>13</td>
</tr>
<tr>
<td>N6</td>
<td>36</td>
<td>39</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2</td>
<td>40</td>
<td>46</td>
<td>36</td>
<td>4</td>
</tr>
<tr>
<td>N3</td>
<td>32</td>
<td>39</td>
<td>37</td>
<td>7</td>
</tr>
<tr>
<td>N5</td>
<td>32</td>
<td>39</td>
<td>28</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2</td>
<td>45</td>
<td>46</td>
<td>36</td>
<td>9</td>
</tr>
<tr>
<td>N5</td>
<td>44</td>
<td>39</td>
<td>28</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2</td>
<td>45</td>
<td>46</td>
<td>46</td>
<td>1</td>
</tr>
</tbody>
</table>

Complexity: $O(S^2m)$ per meta-task, where $m$ is the number of machines and $S$ is the number of tasks in the meta-task.
Conclusion

- The static mapping heuristics tend to be dropped in favor of the dynamic ones due to their advantages.
- The online heuristics are the easiest ones to implement and usually have good results when compared to other dynamic mapping heuristics.
- The batch heuristics are harder to implement, and they also can present the problem of task starvation – need to add an aging mechanism to solve this issue.
- Which dynamic heuristics to use when?
  - On-line when task arrival rate is low (tasks mapped immediately)
  - Batch when task arrival rate is high (group tasks into meta-tasks, then map them)
Conclusion cont’d

- The performance results for the example shown here does not reflect the actual performances of the presented heuristics.

- However, the selected heuristics are among the best performing ones in their categories.

- Mapping on heterogeneous systems is a lot more complex (if done correctly) than on homogenous systems.
References


