Parallelizing with xDSC, a Resource-Constrained Scheduling Algorithm for Shared and Distributed Memory Systems

Dounia KHALDI

CRI, Mathématiques et systèmes
MINES ParisTech

June 19, 2012
Problem Statements

Evolution of the architecture (Multicores, GPUs...)

Evolution of parallel execution environments (OpenMP, MPI, OpenCL...)

Parallel software developed by converting sequential programs by hand
Scheduling ⇒ minimize completion time.

length(path) = communication_cost(edges) + computational_cost(nodes).

Dynamic vs. Static.

List-scheduling heuristics.

```c
in = InitHarris();
//Sobel
SobelX(Gx, in);
SobelY(Gy, in);
//Multiply
MultiplyX(Ixx, Gx, Gx);
MultiplyY(Iyy, Gy, Gy);
MultiplyY(Ixy, Gx, Gy);
//Gauss
Gauss(Sxx, Ixx);
Gauss(Syy, Iyy);
Gauss(Sxy, Ixy);
//Coarsity
Coarsity(out, Sxx, Syy, Sxy);
```
List-Scheduling Processes

- Priorities are computed of all unscheduled nodes:
  - Top level (tlevel(τ)): length of the longest path from the entry node to τ ⇒ earliest possible start-time.
  - Bottom level (blevel(τ)): length of the longest path from τ to the exit node ⇒ latest start-time = CriticalPathLength - blevel(τ).

- The node τ with the highest priority is selected for scheduling.
- τ is allocated to the cluster that offers the earliest start-time.

<table>
<thead>
<tr>
<th>task</th>
<th>tlevel</th>
<th>blevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ4</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>τ3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>τ1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>τ2</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure: A Directed Acyclic Graph
DSC (Dominant Sequence Clustering)  
[Yang and Gerasoulis 1994]

- priority($\tau$) = tlevel($\tau$) + blevel($\tau$).
- A zeroing($\tau_p$, $\tau$) puts $\tau$ in the cluster of a predecessor $\tau_p$ ⇒ reduces tlevel($\tau$) by setting to zero the cost of the incident edge ($\tau_p$, $\tau$).

<table>
<thead>
<tr>
<th>step</th>
<th>task</th>
<th>tlevel</th>
<th>blevel</th>
<th>DS</th>
<th>scheduled tlevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\tau_4$</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>$0^*$</td>
</tr>
<tr>
<td>2</td>
<td>$\tau_3$</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>$2^*$</td>
</tr>
<tr>
<td>3</td>
<td>$\tau_1$</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>$0^*$</td>
</tr>
<tr>
<td>4</td>
<td>$\tau_2$</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>$5^*$</td>
</tr>
</tbody>
</table>

$\kappa_0$ $\kappa_1$

| $\tau_4$ | $\tau_1$ |
| $\tau_3$ | $\tau_2$ |

Figure: A Directed Acyclic Graph
DSC Algorithm Weaknesses

- Number of processors is not predefined $\rightarrow$ blind clustering.
- Memory size is not predefined $\rightarrow$ blind clustering.
- Creates a new cluster when no zeroing is accepted $\rightarrow$ creates long idle slots in already existing clusters.

$\Rightarrow$ xDSC: A MEMORY-CONSTRAINED, NUMBER OF PROCESSOR-BOUNDED EXTENSION OF DSC
Memory Constraint Warranty (MCW):

1. Verifying that the zeroing does not exceed a memory threshold $M$.
2. $\text{task\_data}(\tau)$ is an overapproximation of the amount of memory used by Task $\tau$.
3. $\text{cluster\_data}(k)$ is an overapproximation of the amount of memory used by Cluster $k$.
4. $\text{size\_data}(\text{data\_merge}(\text{cluster\_data}(k), \text{task\_data}(\tau))) \leq M$.

Bounded number of processors:

1. Verifying that new allocations do not exceed a cluster number threshold $P$.
2. $\text{cluster\_time}(k)$ is the start time of the last scheduled task in $k$ plus its task_time.
3. otherwise, $\text{argmin}_{k \in \text{clusters}} \text{cluster\_time}(k)$ under the constraint MCW.
xdSC: Efficient Allocation

Figure: A DAG amenable to cluster minimization

- Allocation of $\tau$ to the last idle slot of $\kappa$, 
- Decreases $\text{tlevel}(\tau)$. 
- For all nodes $\tau_s$ in $\kappa$: 
  - scheduled(successors($\tau_s$)),
  - successors($\tau$) are included in successors($\tau_s$).

<table>
<thead>
<tr>
<th>step</th>
<th>task</th>
<th>t level</th>
<th>b level</th>
<th>DS</th>
<th>tlevel</th>
<th>$\kappa_0$</th>
<th>$\kappa_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\tau_1$</td>
<td>0</td>
<td>15</td>
<td>15</td>
<td>0*</td>
<td>$\tau_1$</td>
<td>$\kappa_1$</td>
</tr>
<tr>
<td>2</td>
<td>$\tau_3$</td>
<td>2</td>
<td>13</td>
<td>15</td>
<td>1*</td>
<td>$\tau_3$</td>
<td>$\kappa_1$</td>
</tr>
<tr>
<td>3</td>
<td>$\tau_2$</td>
<td>2</td>
<td>12</td>
<td>14</td>
<td>3</td>
<td>$\tau_2$</td>
<td>$\kappa_1$</td>
</tr>
<tr>
<td>4</td>
<td>$\tau_4$</td>
<td>8</td>
<td>6</td>
<td>14</td>
<td>7*</td>
<td>$\tau_4$</td>
<td>$\kappa_1$</td>
</tr>
<tr>
<td>5</td>
<td>$\tau_5$</td>
<td>8</td>
<td>5</td>
<td>13</td>
<td>8*</td>
<td>$\tau_5$</td>
<td>$\kappa_1$</td>
</tr>
<tr>
<td>6</td>
<td>$\tau_6$</td>
<td>13</td>
<td>2</td>
<td>15</td>
<td>10*</td>
<td>$\tau_6$</td>
<td>$\kappa_1$</td>
</tr>
</tbody>
</table>
A test (both branches: true + false) constitutes one node (task).
A loop nest is an indivisible node.
A simple instruction is an indivisible node.
⇒ Hierarchy: recursively include KDGs.
Edge Cost, Task Time and Used Data
From Convex polyhedra to Polynomials

1. Edge Cost:
   - Number of bytes of dependences RAW to annotate edges in the KDG,
   - \( \text{edge\_cost}(\tau_i, \tau_j) = \text{size\_data}(\text{regions\_intersection}(\text{read\_regions}(\tau_i), \text{write\_regions}(\tau_j))). \)

2. Task Data:
   - \( \text{task\_data}(\tau) = \text{data\_merge}(\text{read\_regions}(\tau), \text{write\_regions}(\tau)) \)
   - \( \text{data\_merge}(R_1, R_2) = \text{regions\_union}(R_1, R_2) - \text{regions\_intersection}(R_1, R_2) \)

3. Size of regions (convex polyhedra) ⇒ Ehrhart polynomials represent the number of integer points contained in a given parameterized polyhedron.

4. Task Time:
   - An estimation of complexity for each node in the KDG,
   - \( \text{task\_time}(\tau) = \text{complexity\_estimation}(\tau) \Rightarrow \text{Polynomials}. \)
Applications

- Signal processing application ABF (Adaptive Beam Forming) [Griffiths 1969].
- Image processing application Harris corner detector [Harris and Stephens 1988]: detect the point of interest in an image.

Machines

- SMP: 2-socket AMD quadcore Opteron with 8 cores, \( M = 16 \text{Gb of RAM}, 2.4 \text{GHz} \).
- DMP: 6 bicomputer processors Intel(R) Xeon(R), \( M = 32 \text{Gb of RAM per processor}, 2.5 \text{GHz} \).
When data are known numerical parameters, then each task polynomial is a constant (case of the application ABF).

However, when input data are unknown at compile time (case of the application Harris), we use a simple heuristic to check the behavior of that polynomials, by comparing the coefficients of their monomials.

Assume that all polynomials are monomials on the same bases.

<table>
<thead>
<tr>
<th>Function</th>
<th>Complexity (polynomial)</th>
<th>Static time estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitHarris</td>
<td>9 \times \text{sizeN} \times \text{sizeM}</td>
<td>9</td>
</tr>
<tr>
<td>SobelX</td>
<td>60 \times \text{sizeN} \times \text{sizeM}</td>
<td>60</td>
</tr>
<tr>
<td>SobelY</td>
<td>60 \times \text{sizeN} \times \text{sizeM}</td>
<td>60</td>
</tr>
<tr>
<td>MultiplyY</td>
<td>20 \times \text{sizeN} \times \text{sizeM}</td>
<td>20</td>
</tr>
<tr>
<td>Gauss</td>
<td>85 \times \text{sizeN} \times \text{sizeM}</td>
<td>85</td>
</tr>
<tr>
<td>CoarsitY</td>
<td>34 \times \text{sizeN} \times \text{sizeM}</td>
<td>34</td>
</tr>
<tr>
<td>One image transfer</td>
<td>4 \times \text{sizeN} \times \text{sizeM}</td>
<td>4</td>
</tr>
</tbody>
</table>
FILE *finstrumented = fopen("instrumented_equake.in","w");
...
fprintf(finstrumented, "62\n", 179 * ARCHelems + 3);
for (i = 0; i < ARCHelems; i++){
    for (j = 0; j < 4; j++)
        cor[j] = ARCHvertex[i][j];
}
...
fprintf(finstrumented, "163\n", 20 * ARCHnodes + 3);
for (i = 0; i <= ARCHnodes-1; i += 1)
    for (j = 0; j <= 2; j += 1)
        disp[disptplus][i][j] = 0.0;
...
Experiments: ABF and equake

**Figure:** OpenMP/MPI vs. sequential speedup (ABF)

**Figure:** OpenMP/MPI vs. sequential speedup (equake)
Experiments: Harris
Conclusion

- xDSC: a new static scheduling,
- Precise and efficient cost model,
- Targeting both shared and distributed memory architectures,
- Memory constraint, Bounded number of processors, Efficient processor allocation.

Future Work

- Automatic code generation: OpenMP + MPI.
- Efficient hierarchical processor allocation strategy in order to yield a better xDSC-based parallelization process
Parallelizing with xDSC, a Resource-Constrained Scheduling Algorithm for Shared and Distributed Memory Systems

Dounia KHALDI

CRI, Mathématiques et systèmes
MINES ParisTech

June 19, 2012