Tools for porting programs to a new parallel programming language

(Feb. 3, 2011)

Mohsen Vakilian  (mvakili2@illinois.edu)
(Joint work with Rob Bocchino, Danny Dig,
Jeff Overbey, Vikram Adve, Ralph Johnson)

port using automated tools

sequential program in a base language \( X \)

parallel program in a target language \( X' \) that is based on \( X \)
Some parallel programming environments are based on sequential ones.

DPJ

Intel Array Building Block

Microsoft PLINQ
PLINQ extends LINQ

```csharp
IEnumerable<T> data = ...;
var q = data.AsParallel().Where(x => p(x)).
    Orderby(x => k(x)).
    Select(x => f(x));
foreach (var e in q) a(e);
```
DPJ has region, RPL, region parameter and effects.

class Node<region P> {
    region L, R;
    double mass in P;
    Node<region P:L> left in P:L;
    Node<region P:R> right in P:R;

    void setMassForTree(double mass) {
        writes P, P:L:* , P:R:* {
           this.mass = mass;
           if (left != null) left.setMassForTree(mass);
           if (right != null) right.setMassForTree(mass);
        }
    }
}
It is difficult for programmers or the compiler to infer the effects.

constraint graph
DPJizer infers effects better than programmers.

writes $P, P:L:*$, instead of $P:R:*$

writes $P$ instead of $P$
class Tree {
    Body<\langle \frac{[i]}{\pi_1} \rangle >[] # \langle \frac{i}{i} \rangle bodies;
    
    void computeGrav() {
        foreach (int j in Ø, bodies.length) {
            HGStruct<\langle \frac{[j]}{\pi_2} \rangle> hg = new HGStruct<\langle \frac{[j]}{\pi_3} \rangle>();
            bodies[j].hackgrav(hg);
        }
    }
}

class Body<region BR> {
    Vector<\langle \frac{BR}{\pi_4} \rangle> vel in BR = new Vector<\langle \frac{BR}{\pi_5} \rangle>();
    
    void hackgrav(HGStruct<\langle \frac{BR}{\pi_6} \rangle> hg) {
        vel.<\langle \frac{\pi_11}{\pi_11} \rangle> SETV(hg.acc * 0);
    }
}
class Vector<region VR> {
    double[] \( \frac{VR}{\pi_7} \) \# \( \frac{k}{k} \) elts in VR = new double[3] \( \frac{VR}{\pi_8} \) \# \( \frac{l}{l} \);

    <region SR> void SETV(Vector<region SR> u) {
        for (int m = 0; m < 3; m++) {
            elts[m] = u.elts[m];
        }
    }

}  

class HGStruct<region HR> {
    Vector<region HR> acc 0 in HR;
}

Region inclusion constraints are derived from subtyping rules.

<table>
<thead>
<tr>
<th>Type Definition</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>double[3]</code></td>
<td><code># k els in VR = VR[VR ← π₁₀][HR ← π₆] ≤ VR[VR ← π₉][VR ← π₄][SR ← π₁₁]</code></td>
</tr>
<tr>
<td><code>vel.(π₁₁)</code></td>
<td><code>HR[HR ← π₂] ≤ HR[HR ← π₆][BR ← π₁][i ← j]</code></td>
</tr>
<tr>
<td><code>bodies[j].hackgrav(hg)</code></td>
<td><code>π₃ ≤ π₂</code></td>
</tr>
<tr>
<td><code>HGStruct(π₂)hg = new HGStruct(π₃)</code></td>
<td><code>π₅ ≤ π₄</code></td>
</tr>
<tr>
<td><code>Vector(π₄) vel in BR = new Vector(π₅)</code></td>
<td><code>π₈ ≤ π₇</code></td>
</tr>
</tbody>
</table>
Parallel statements generate disjointness constraints.

```
foreach (int j in 0, bodies.length) {
    HGStruct<π2> hg = new HGStruct<π3>();
    bodies[j].hackgrav(hg);
}
```

\[ E(\text{hackgrav})[BR\leftarrow \pi_1][i\leftarrow j][j\leftarrow a] \]

\# \[ E(\text{hackgrav})[BR\leftarrow \pi_1][i\leftarrow j][j\leftarrow b] \]
Write effects should be disjoint from each other, and read effects should be disjoint from write effects.

**foreach loop in computegrav**

- invokes with $[BR \leftarrow \pi_1][i \leftarrow j]$

  **hackgrav**

  - reads $BR, \pi_6$

    - invokes with $[VR \leftarrow \pi_4][SR \leftarrow \pi_{11}]$

    **SETV**

    - reads $VR, \pi_9, \pi_7[VR \leftarrow \pi_9][k \leftarrow ?]$

    - writes $\pi_7[k \leftarrow ?]$
The index variable of the foreach loop should differentiate the write effects.

\[ \text{writes } \pi_7[k \leftarrow ?][v_r \leftarrow \pi_4][s_r \leftarrow \pi_{11}][b_r \leftarrow \pi_1[i \leftarrow j] \]

\[ \pi_7 = v_r \]
\[ \pi_4 = b_r \]
\[ \pi_1 = [i] \]
Read effects should be disjoint from write effects.

reads $\pi_9[VR \leftarrow \pi_4][SR \leftarrow \pi_{11}][BR \leftarrow \pi_i][i \leftarrow j]$

$\pi_9 = VR$ OR $\pi_9 = SR, \pi_{11} = BR$
Region inclusion constraints determine the rest of variables.

\[ \pi_1 = [i], \ pi_4 = BR, \ pi_7 = VR, \ pi_9 = SR, \ pi_{11} = BR \]

\[ \pi_8 \subseteq \pi_7 \]

\[ VR[VR \leftarrow \pi_{10}][HR \leftarrow \pi_6] \subseteq VR[VR \leftarrow \pi_9][VR \leftarrow \pi_4][SR \leftarrow \pi_{11}] \]

\[ HR[HR \leftarrow \pi_2] \subseteq HR[HR \leftarrow \pi_6][BR \leftarrow \pi_1][i \leftarrow j] \]

\[ \pi_3 \subseteq \pi_2 \]

\[ \pi_5 \subseteq \pi_4 \]

\[ \pi_8 = VR \]

\[ \pi_6 = BR, \ pi_{10} = HR \]

\[ \pi_2 = [j] \]

\[ \pi_3 = [j] \]

\[ \pi_5 = BR \]
This work is funded by Microsoft and Intel through UPCRC and by NSF grants.

The DPJ homepage is http://dpj.cs.illinois.edu