Parallelizing RRT on Distributed-Memory Architectures

Didier Devaurs, Thierry Siméon, Juan Cortés
{devaurs, nic, jcortes}@laas.fr

Toulouse, France
✓ Popular sampling-based planning algorithm

✓ RRT applied in
  • robot motion planning (with holonomic, non-holonomic, kinodynamic, or kinematic-closure constraints)
  • validation and control of hybrid systems
  • molecular simulation, for the analysis of
    - genetic regulatory network dynamics
    - protein-ligand interactions
  • etc.
An example in structural biology
Such problems are computationally expensive
  • many dimensions
  • complex geometric constraints

Some improvements already proposed for RRT
  • using efficient nearest neighbour search [Yershova07]
  • controlling the sampling domain [Jaillet05]
  • employing gap reduction techniques [Cheng04]

Our (complementary) approach
  • exploiting speedup from parallel computation
Existing work
- mainly limited to shared-memory architectures
- small-scale parallelism

Distributed-memory architectures
- offer large-scale parallelism
- require using the message-passing paradigm

We present
- three parallel versions of RRT
- an evaluation on several motion planning problems
Three parallelization schemes

✓ Several RRTs are built concurrently
  • **OR parallel RRT**
    - Each process computes its own RRT, and
    - the first to finish broadcasts a termination message
    - Gain achieved by finding a small-sized solution
    - Straightforward implementation

✓ One RRT is built collaboratively
  • **Distributed RRT**
  • **Manager-worker RRT**
Algorithm 1: OR parallel RRT

**input**: the configuration space $C$, the root $q_{init}$
**output**: the tree $T$

1. $T \leftarrow \text{initTree}(q_{init})$
2. while not stopCondition($T$) or received($endMsg$) do
   3. $q_{rand} \leftarrow \text{sampleRandomConfiguration}(C)$
   4. $q_{near} \leftarrow \text{findBestNeighbor}(T, q_{rand})$
   5. $q_{new} \leftarrow \text{extend}(q_{near}, q_{rand})$
   6. if not tooSimilar($q_{near}, q_{new}$) then
      7. addNewNodeAndEdge($T$, $q_{near}$, $q_{new}$)

8. if stopCondition($T$) then
9. broadcast($endMsg$)
Partition the global task into sub-tasks assigned to several processes

**Distributed RRT**: exploratory decomposition
- Each process performs its own sampling of the search space
- Each process has its own copy of the tree, and
- exchanges new nodes with the other processes
- Coordination reduced to: termination detection

**Manager-worker RRT**: functional decomposition
- Its implementation requires more work than Distributed RRT
Algorithm 2: Distributed RRT

\textbf{input}: the configuration space $C$, the root $q_{init}$
\textbf{output}: the tree $T$

1 $T \leftarrow \text{initTree}(q_{init})$

2 \textbf{while not} stopCondition($T$) or received($endMsg$) \textbf{do}

3 \hspace{1em} \textbf{while} received(nodeData($q_{new}$, $q_{near}$)) \textbf{do}

4 \hspace{2em} \text{addNewNodeAndEdge}($T$, $q_{near}$, $q_{new}$)

5 \hspace{1em} $q_{rand} \leftarrow \text{sampleRandomConfiguration}(C)$

6 \hspace{1em} $q_{near} \leftarrow \text{findBestNeighbor}($$T$, $q_{rand}$)

7 \hspace{1em} $q_{new} \leftarrow \text{extend}($$q_{near}$, $q_{rand}$)

8 \hspace{1em} \textbf{if not} tooSimilar($q_{near}$, $q_{new}$) \textbf{then}

9 \hspace{2em} \text{addNewNodeAndEdge}($T$, $q_{near}$, $q_{new}$)

10 \hspace{2em} \text{broadcast(nodeData($q_{new}$, $q_{near}$))}

11 \hspace{1em} \textbf{if} stopCondition($T$) \textbf{then}

12 \hspace{2em} \text{broadcast($endMsg$)}
Manager-worker RRT (or Master-slave RRT)

- Functional decomposition of the sub-tasks
- Dynamic and centralized task-scheduling strategy
- Manager
  - maintains the tree (add new nodes and edges)
  - samples random configurations
  - performs the nearest neighbour search
  - evaluates the stopping conditions
- Workers
  - have no copy of the tree
  - perform extension attempts
Algorithm 3: Manager-worker RRT

\textbf{input} : the configuration space $C$, the root $q_{init}$
\textbf{output} : the tree $T$

1. \textbf{if} $processID = mgr$ \textbf{then}
   2. \quad $T \leftarrow initTree(q_{init})$
   3. \quad \textbf{while} not stopCondition($T$) \textbf{do}
   4. \quad \quad \textbf{while} received($nodeData(q_{new}, q_{near})$) \textbf{do}
   5. \quad \quad \quad addNewNodeAndEdge($T$, $q_{near}$, $q_{new}$)
   6. \quad \quad $q_{rand} \leftarrow sampleRandomConfiguration(C)$
   7. \quad \quad $q_{near} \leftarrow findBestNeighbor(T, q_{rand})$
   8. \quad \quad $w \leftarrow chooseWorker()$
   9. \quad \quad send($expansionData(q_{rand}, q_{near}), w$)
   10. \quad broadcast($endMsg$)
11. \textbf{else}
12. \quad \textbf{while} not received($endMsg$) \textbf{do}
13. \quad \quad receive($expansionData(q_{rand}, q_{near}), mgr$)
14. \quad \quad $q_{new} \leftarrow extend(q_{near}, q_{rand})$
15. \quad \quad \textbf{if} not tooSimilar($q_{near}, q_{new}$) \textbf{then}
16. \quad \quad \quad send($nodeData(q_{new}, q_{near}), mgr$)
Three molecular simulation problems

- Free-flying objects (6 DOFs)
- Different configuration-space topologies
  - GAB: weak geometrical constraints (but long distance)
  - BCL: medium geometrical constraints
  - CALB: strong geometrical constraints

<table>
<thead>
<tr>
<th>Problem name</th>
<th>BCL</th>
<th>CALB</th>
<th>GAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem type</td>
<td>![Diagram]</td>
<td>![Diagram]</td>
<td>![Diagram]</td>
</tr>
<tr>
<td>Sequential RRT</td>
<td>$T_s$ (s)</td>
<td>1.4 ± 0.81</td>
<td>148 ± 129</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>46 ± 19</td>
<td>1629 ± 1365</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>821 ± 474</td>
<td>81023 ± 69917</td>
</tr>
</tbody>
</table>
Speedup: $S(p) = \frac{T_S}{T_P(p)}$
- $T_S$ = sequential runtime on 1 processor
- $T_P(p)$ = parallel runtime on $p$ processors
- measures the performance gain achieved on $p$ processors

Efficiency: $E(p) = \frac{S(p)}{p}$
- evaluates the ratio of resources being used

Scalability
- evolution of $S(p)$ with respect to $p$
OR parallel RRT

- good scalability on CALB only
  (i.e. on problems with a great variability in sequential runtime)
Distributed RRT

- good scalability on all problems
- no speedup decrease is observed (i.e. computational gain always dominates over communication costs)

<table>
<thead>
<tr>
<th>Problem type</th>
<th>BCL</th>
<th>CALB</th>
<th>GAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential RRT</td>
<td>Ts (s)</td>
<td>1.4 ± 0.81</td>
<td>148 ± 129</td>
</tr>
<tr>
<td>N</td>
<td>46 ± 19</td>
<td>1629 ± 1365</td>
<td>615 ± 90</td>
</tr>
<tr>
<td>E</td>
<td>821 ± 474</td>
<td>81023 ± 69917</td>
<td>770 ± 121</td>
</tr>
</tbody>
</table>
Manager-worker RRT

- poor scalability on all problems
- communication costs outweigh computational gain

<table>
<thead>
<tr>
<th>Problem name</th>
<th>BCL</th>
<th>CALB</th>
<th>GAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem type</td>
<td><img src="image1" alt="Graph" /></td>
<td><img src="image2" alt="Graph" /></td>
<td><img src="image3" alt="Graph" /></td>
</tr>
<tr>
<td>Sequential RRT</td>
<td><img src="image4" alt="Table" /></td>
<td><img src="image5" alt="Table" /></td>
<td><img src="image6" alt="Table" /></td>
</tr>
<tr>
<td>Ts (s)</td>
<td>1.4 ± 0.81</td>
<td>148 ± 129</td>
<td>62 ± 12</td>
</tr>
<tr>
<td>N</td>
<td>46 ± 19</td>
<td>1629 ± 1365</td>
<td>615 ± 90</td>
</tr>
<tr>
<td>E</td>
<td>821 ± 474</td>
<td>81023 ± 69917</td>
<td>770 ± 121</td>
</tr>
</tbody>
</table>
In our settings, the cost of an RRT extension is low
  • new nodes created by linear interpolation
  • motion validation limited to collision detection

What happens when computational costs are higher?

Controlled experiment in which the extension cost is artificially increased
  • repeat I times the collision detection routine
  • evaluate the evolution of the speedup in relation to I
  • tests performed on BCL, on 8, 16 and 32 processors
**Efficiency evolution (32 processors)**

- **OR parallel RRT**
  - no improvement (as expected)
- **Distributed RRT**
  - small increase in speedup
- **Manager-worker RRT**
  - dramatic increase in speedup
  - almost **optimal efficiency** (0.9) achieved at some point
  - then the manager becomes a bottleneck & speedup decreases
- **Why this overall increase in speedup?**
  - when \( I \) increases, communication load becomes insignificant compared to computational load
✓ OR parallel RRT (the simplest one)
   + good speedup when runtime variability is great
   - poor speedup otherwise & no possible scaling in efficiency
   - memory scalability issue (each process builds its own tree)

✓ Distributed RRT (the most consistent one)
   + good speedup in general
   - efficiency does not scale well w.r.t. computational costs
   - memory scalability issue (replicated tree)

✓ Manager-worker RRT (best for extension-expensive problems)
   + efficiency scales well w.r.t. computational costs
   + very good speedup when the RRT extension is expensive
   - poor speedup otherwise & the manager can become a bottleneck
Future work

✅ Tests on real-life extension-expensive problems
  • Molecular motion planning involving potential energy computation in the extension step
  • Does Manager-worker RRT perform best in that context?

✅ Improvement of the current parallel algorithms
  • Manager-worker RRT
    - use several managers to limit the bottleneck effect
  • Distributed RRT
    - combine the multi-threading and message-passing approaches to mitigate the memory scalability issue
  • Develop a combination of the three algorithms

✅ Development of finer-grained parallel versions of RRT
  • Parallelization of the nearest neighbour search, etc.