Speeding Up Learning in Real-time Search through Parallel Computing

Vinicius Marques Terra
Luiz Chaimowicz
Renato Ferreira

Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

October 28, 2011

Contextualization

Real-time search

- Traditional heuristic search (e.g., A*): plan all path, then execute:

- Real-time search: doesn't compromise time restrictions;
- fast response regardless the problem size;
- Interleave planning (in limited area) and plan execution:

Real-time search

Learning step

- Only put a bound on search depth doesn't guarantee optimal solution;
- Real-time search must also be able to deal with dynamic environments;
- *Learning step*: refines the heuristic values for visited states;
- Learning occurs between plan and action;

Convergence process:

"When the same planning task is solved repeatedly, the learning acquired ensures the convergence to the optimal path." (R. E. Korf - LRTA*, 1990)

Real-time search

The problem

- Perform search/learn in a limited area due to real-time constraints makes the run to convergence a lengthy process;
- **How learning can be accelerated so that fewer searches/trials are performed until the optimal path?**
- Our approach - parallelism to speedup the convergence process;

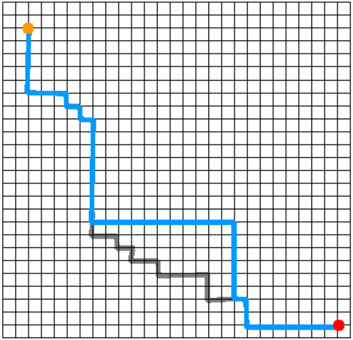
Previous attempts on speeding up convergence

Some relevant works

- FALCONS (Fast Learning and Converging Search) - proposes alternatively way to select successors, showing its influence on convergence speed;
- LRTA*(k) - alternate strategy to propagate learn: update heuristic estimates of up to k states;
- Local Search Space LRTA* (LSS-LRTA*) - uses bounded A* to search, and Dijkstra on visited states to learn;

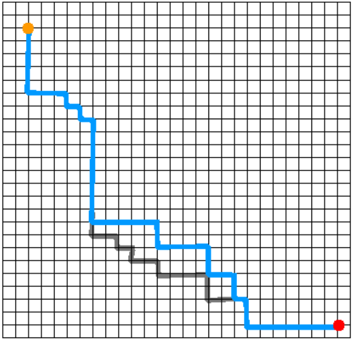
How our work differs from others

- Those works and many others focus on modifying the LRTA* original structure to reduce convergence time;
- Strictly sequential behavior;
- We introduce the paradigm of parallel programming on the convergence:



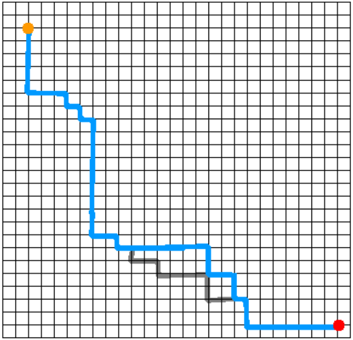
How our work differs from others

- Those works and many others focus on modifying the LRTA* original structure to reduce convergence time;
- Strictly sequential behavior;
- We introduce the paradigm of parallel programming on the convergence:



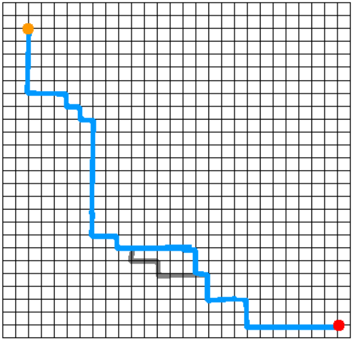
How our work differs from others

- Those works and many others focus on modifying the LRTA* original structure to reduce convergence time;
- Strictly sequential behavior;
- We introduce the paradigm of parallel programming on the convergence:



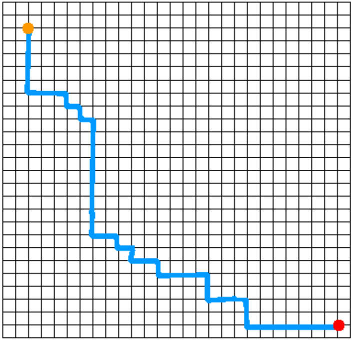
How our work differs from others

- Those works and many others focus on modifying the LRTA* original structure to reduce convergence time;
- Strictly sequential behavior;
- We introduce the paradigm of parallel programming on the convergence:



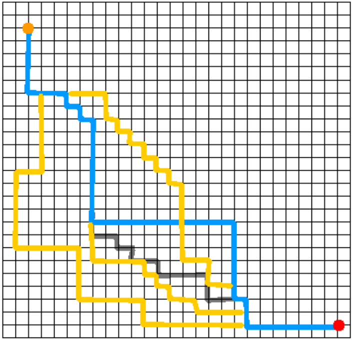
How our work differs from others

- Those works and many others focus on modifying the LRTA* original structure to reduce convergence time;
- Strictly sequential behavior;
- We introduce the paradigm of parallel programming on the convergence:



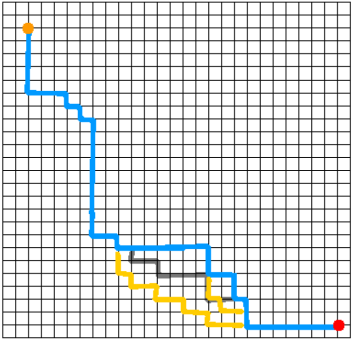
How our work differs from others

- Those works and many others focus on modifying the LRTA* original structure to reduce convergence time;
- Strictly sequential behavior;
- We introduce the paradigm of parallel programming on the convergence:



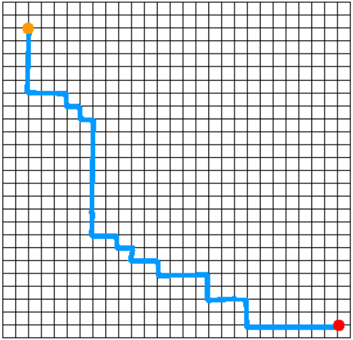
How our work differs from others

- Those works and many others focus on modifying the LRTA* original structure to reduce convergence time;
- Strictly sequential behavior;
- We introduce the paradigm of parallel programming on the convergence:



How our work differs from others

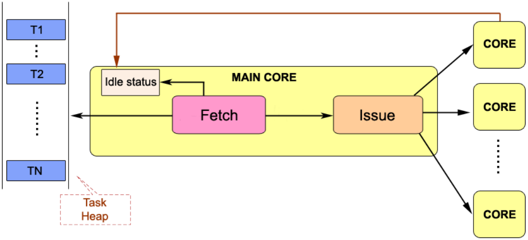
- Those works and many others focus on modifying the LRTA* original structure to reduce convergence time;
- Strictly sequential behavior;
- We introduce the paradigm of parallel programming on the convergence:



Parallelization method

Description

- execution flow - based on master/slave protocol:

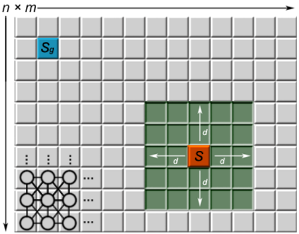


- master also has execute the main search under real-time restrictions;
- search performed by auxiliary cores doesn't require real-time constraints;

Parallelization method

Description

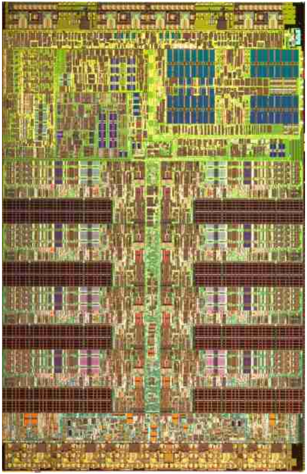
- Parallelization designed considering a *distributed memory system*;
- Hash → matrix;
- Data parallelism - task composed by:
  - tile of the map - states on the graph;
  - current/start and goal points;
  - heuristic values matching map tile states - learning acquired;



Cell Broadband Engine

Architecture used to implement the parallelization

- Heterogeneous Multiprocessing - 9 Core Processor;
- **PPE** - Power Processor Element (PowerPC 64-bits);
- **SPE** - Synergistic Processing Element (SIMD RISC 256KB LS);
- **EIB** - Element Interconnect Bus (DMA);



Cell Broadband Engine

Implementation issues

- SPEs limited LS: code+data structures+execution stack on *256KB*;
- No cache: hide DMA latency with double buffering;

Important consideration

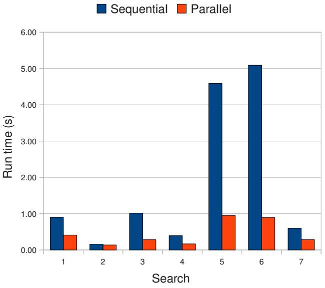
Despite the Cell singular details, the parallelization proposed here fits any architecture;

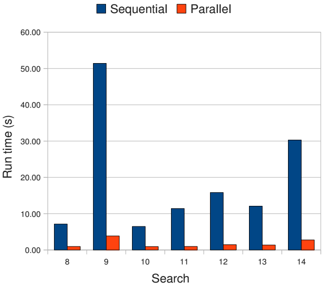
Experimental Results

Overall info

- LSS-LRTA* with *lookahead* up to 3$\times$ higher on the auxiliary cores;
- 14 random searches performed among real game maps;
- 7 with minor convergence time and 7 with longer convergence on sequential run;

Total Time to convergence



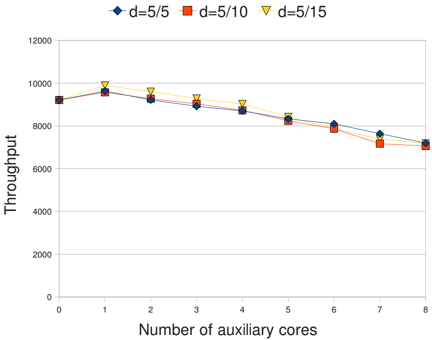


Section\Run	1	2	3	4	5	6	7
Processing	0.1678	0.0553	0.1336	0.0690	0.5614	0.5922	0.1219
Synchronization	0.2421	0.0829	0.1475	0.0984	0.3852	0.2977	0.1608
Management	0.0008	0.0003	0.0006	0.0004	0.0015	0.0024	0.0006
Total	**0.4107**	**0.1385**	**0.2817**	**0.1677**	**0.9481**	**0.8923**	**0.2833**
Sequential	**0.9010**	**0.1598**	**1.0130**	**0.3941**	**4.5859**	**5.0894**	**0.5975**

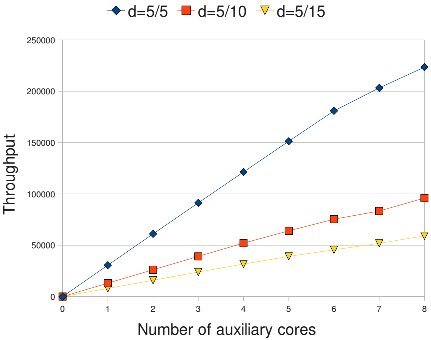
Section\Run	8	9	10	11	12	13	14
Processing	0.527	2.709	0.593	0.660	1.024	0.984	1.874
Synchronization	0.434	1.138	0.344	0.308	0.420	0.380	0.884
Management	0.003	0.015	0.003	0.004	0.005	0.006	0.007
Total	**0.964**	**3.862**	**0.940**	**0.971**	**1.449**	**1.369**	**2.765**
Sequential	**7.150**	**51.410**	**6.480**	**11.420**	**15.836**	**12.050**	**30.280**

Throughput

Main core



Auxiliary cores



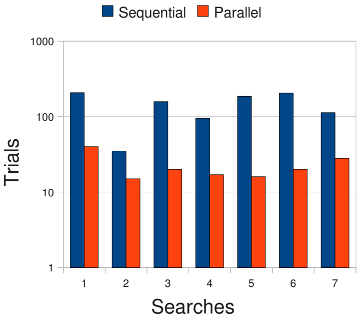
Game execution scenario

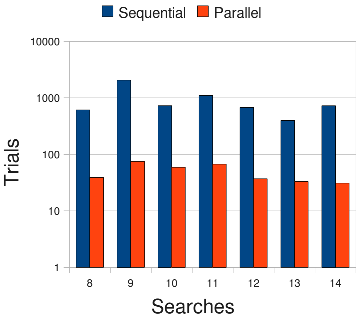
Description of experiment

- Previous experiments: uninterrupted and successive calls for plan step;
- Game scenario: *game-loop* contains other kind of processing tasks;
- One loop per frame - 60fps - 1/60s per loop;
- With 1 call for plan step per *game-loop*: 1/60s between calls;

Game execution scenario

Trials to convergence[1]





Method\Run	1	2	3	4	5	6	7
Sequential	208	35	158	95	186	205	113
Parallel	34	13	22	12	9	7	15

Method\Run	8	9	10	11	12	13	14
Sequential	611	2058	727	1098	675	397	726
Parallel	13	20	26	26	8	7	5

[1] graphics are on logarithmic scale

Final considerations

- Generated overhead may be compensated by smaller *lookaheads* without compromising gains in terms of convergence;
- A *lookahead* 3x higher for the aux searches: average gains one order of magnitude higher than sequential;
- Game execution scenario: background executions in the aux cores significantly reduce convergence time;

Questions?