Large-Scale Parallel Best-First Search for Optimal Planning

Akihiro Kishimoto

Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Japan

kishimoto@is.titech.ac.jp

Joint work with Alex Fukunaga and Adi Botea

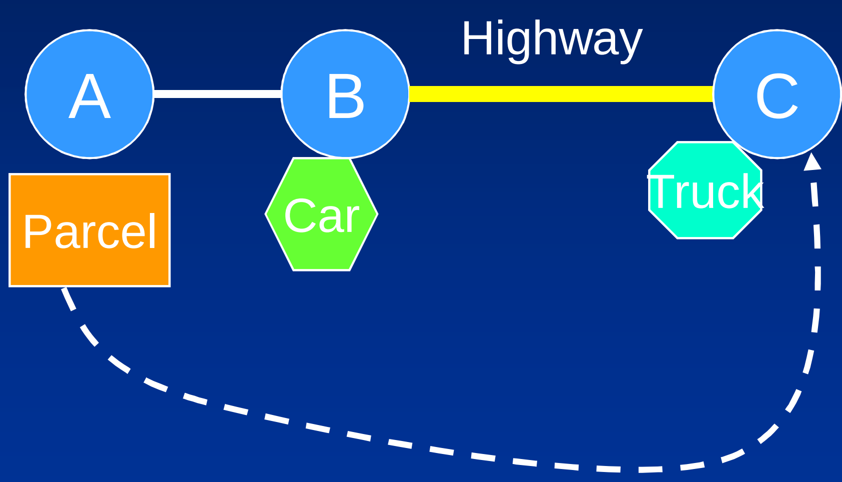
Overview

- Research Motivations
- Overview of Sequential Planning System
- Issues on Parallel Search
- Hash Distributed A*
- Experimental Results
- Conclusions and Future Work

What Is Planning?

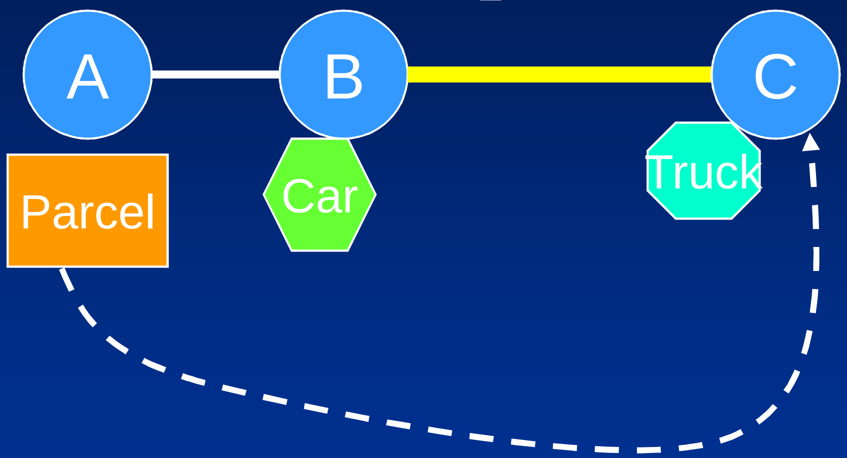
- Represent problem and goal
  - STRIPS, PDDL, etc
- Returns plan to achieve goal
- Representative subject of AI research
- Can be used for many applications
  - Spacecraft, autonomous robots, manufacturability analysis, games [Ghallab:2004]

Example of Optimal Planning: Transportation Planning Task



1. Move-Car B->A
2. Load-Car-with-Parcel@A
3. Move-Car A->B
4. Unload-Parcel-from-Car@B
5. Move-Truck C->B
6. Load-Truck-with-Parcel@B
7. Move-Truck B->C
8. Unload-Parcel-from-Truck@C

SAS+ Representation [Bäckström:1992]



- State S=<Parcel, Car,Truck>
- Parcel ∈ {@A, @B, @C, in-Car, in-Truck}
- Car ∈ {@A,@B}
- Truck ∈ {@B,@C}

Initial State: <Parcel=@A, Car=@B, Truck=@C>

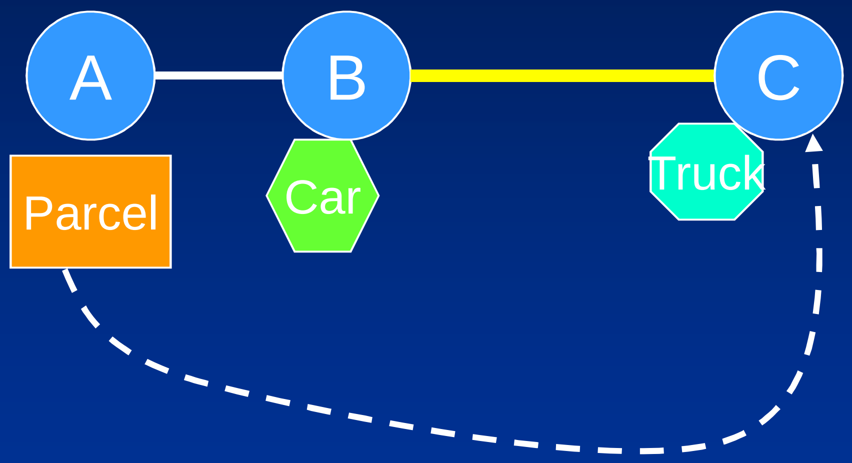
Goal: <Parcel=@C>

Example of Operators in SAS+

- Move-Car B->A

  PRE=<Car=@B>
  EFF=<Car=@A>

- …



Example of applying Move Car B->A

State = <Parcel=@A, Car=@B, Truck=@C>

New State = <Parcel=@A,Car=@A,Truck=@C>

Optimal Planning and Search

- Optimal planning is simpler but still difficult
- Planning problems can be solved by conducting search
  - State = search node, operator = branch
- Search is incorporated into high-performance planning systems (*planners*)
- Search requires intensive computation
  - Combinatorial complexity of search space
  - Necessity of real-time response

Marriage of Parallel Computing and Optimal Planning

- *Parallel search* is an important way to scale up planners
  - Search requires intensive computation
  - Parallel computing becomes ubiquitous
  - Parallel computing provides CPU and memory resources to solve hard problems
  - Speed of individual CPU core doesn’t increase as rapidly as in past decades

Overview of the Fast Downward Planner [Helmert:2007]

- One of the best sequential planners
- Use SAS+ to represent problem
  – states, operators, initial state, and goal
- Generate heuristics automatically
  – Admissible and consistent heuristic
- Perform A* search and return optimal solution

A* Search

- Best-first search using OPEN and CLOSED lists
  - OPEN maintains nodes not expanded yet
  - CLOSED maintains nodes already expanded
    - Used to find a path from root to goal
    - Detect DAG to avoid duplicate search effort
  - Dequeue and expand best node *n* in OPEN
  - Save *n*'s successors to OPEN and *n* to CLOSED
  - Continue until finding an optimal solution
- Requires a large amount of memory

Obstacles to Parallel A* Search

How to distribute work by avoiding overhead?

- Search overhead
  - Extra states explored by parallel search
- Synchronization overhead
  - Idle time wasted at synchronization points
- Communication overhead
  - Extra cost caused by information exchange

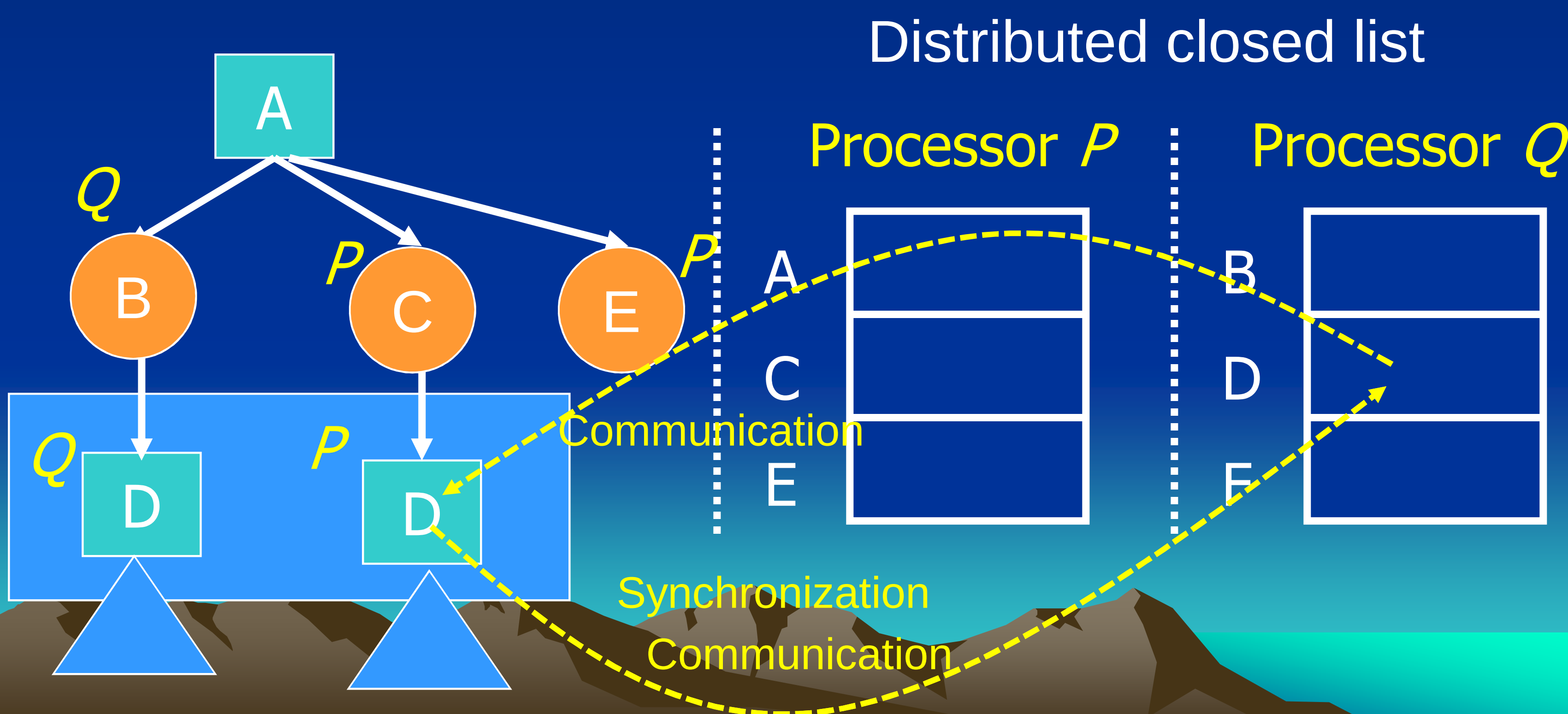
These overheads depend on one another

Work-Stealing – Traditional Approach to Balance Workload

- Each processor places work on its local work queue
- Idle processor randomly selects a “victim” processor to steal work from
- Work-stealing tries to evenly allocate work
- Work-stealing is most popular in shared-memory environments

Issues on Parallel A* Search in Directed Graph

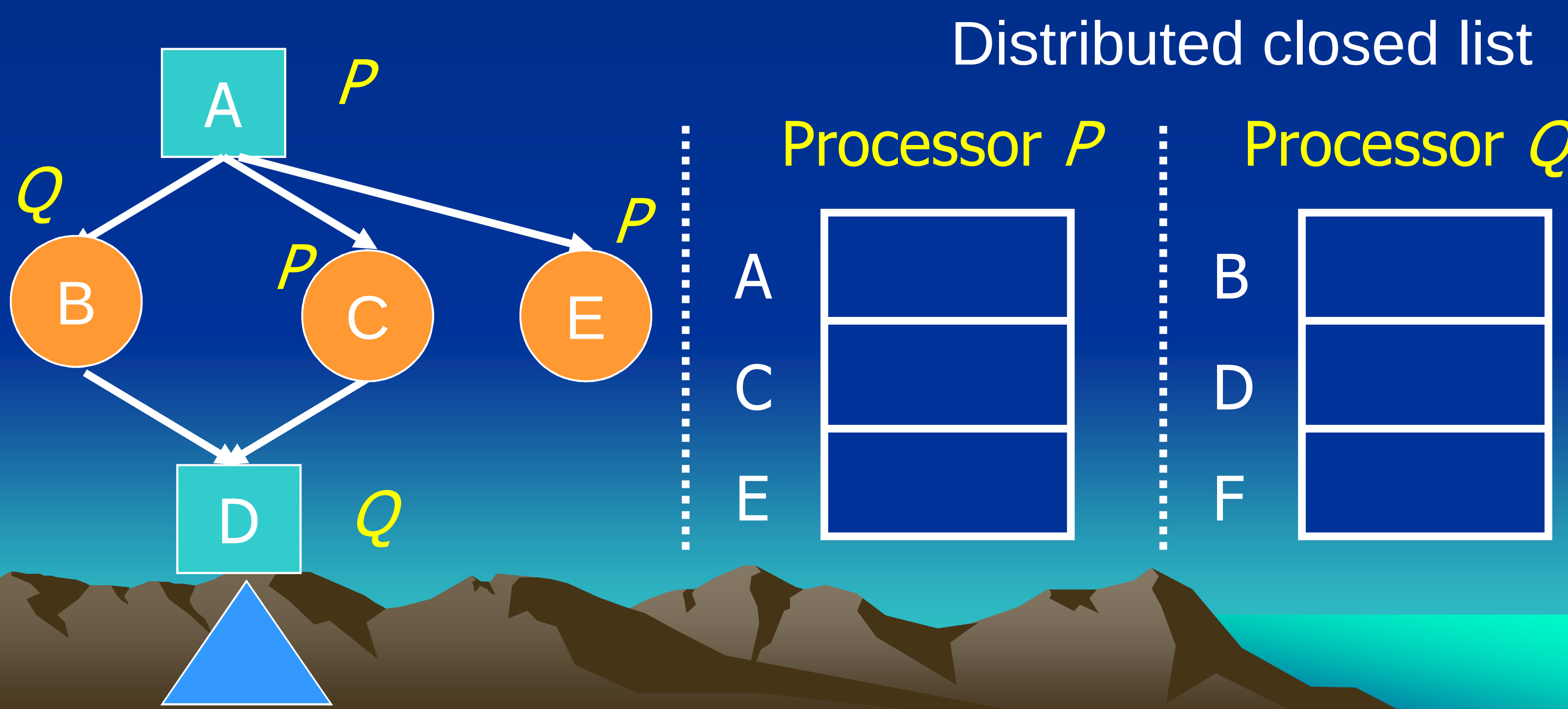
- Search space of many planning problems is not tree, but DAG/DCG



Hash Distributed A* (HDA*)
[Kishimoto, Fukunaga & Botea:ICAPS2009]

- Move work where data is located

[Romein et al.:AAAI1999] [Kishimoto & Schaeffer:ICPP2002]



Advantages of HDA* (1/2)

- Duplicate search can be detected
- More memory is available for OPEN and CLOSED lists
- Work distribution is almost uniform
  - Zobrist function [Zobrist:1970]
    - Hash key computed by using pre-computed random table
- Communication overhead is not an issue
  - Several states can be packed into one message to send

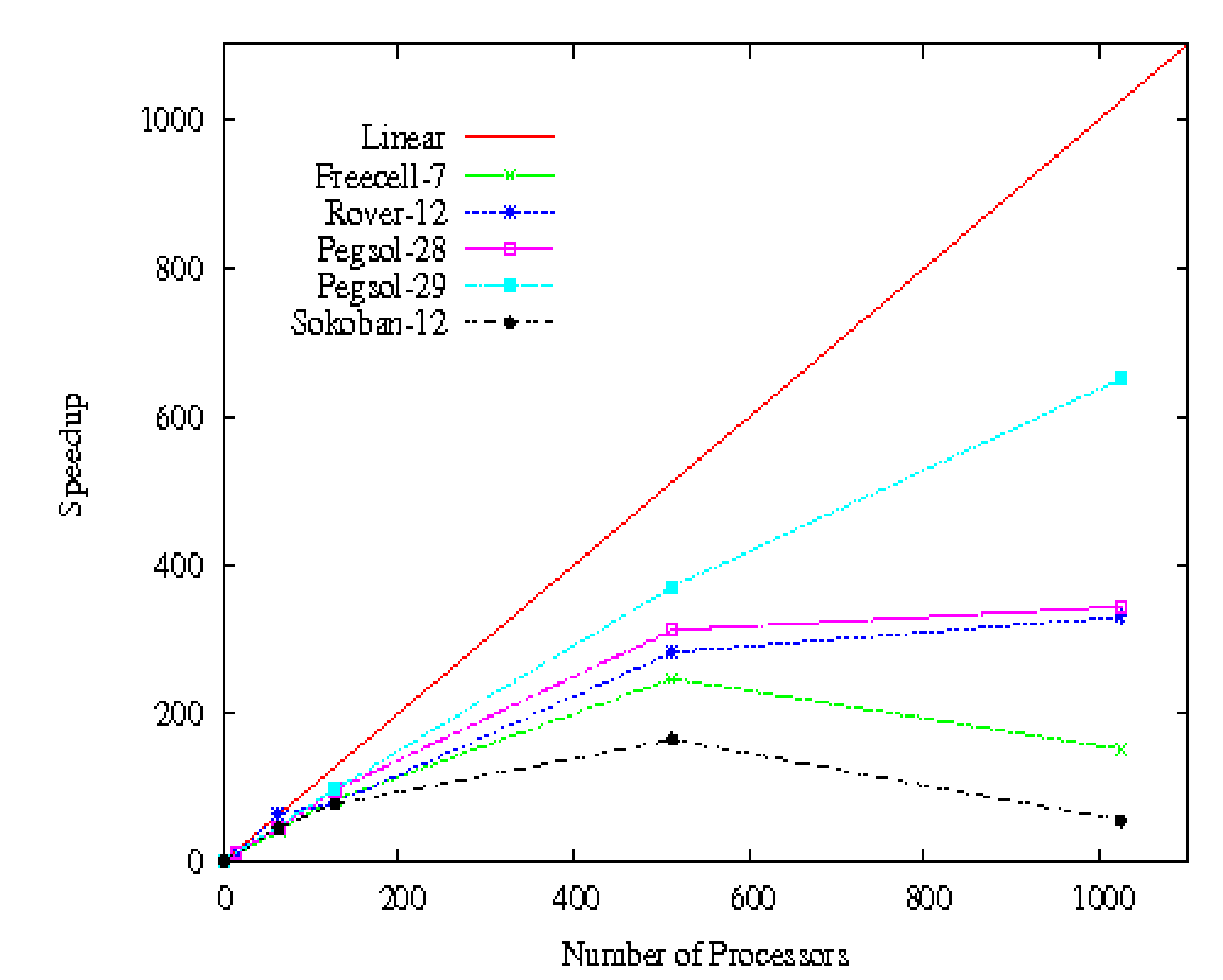
Advantages of HDA* (2/2)

- *Asynchronous communication* is key feature
  - Can work on next node immediately after sending out work to destination
  - PRA*[Evett:1995] is also parallel A* search distributing work based on hash keys
    - *Synchronous*
    - Can be *slower* than sequential search [Burns:IJCAI2009]

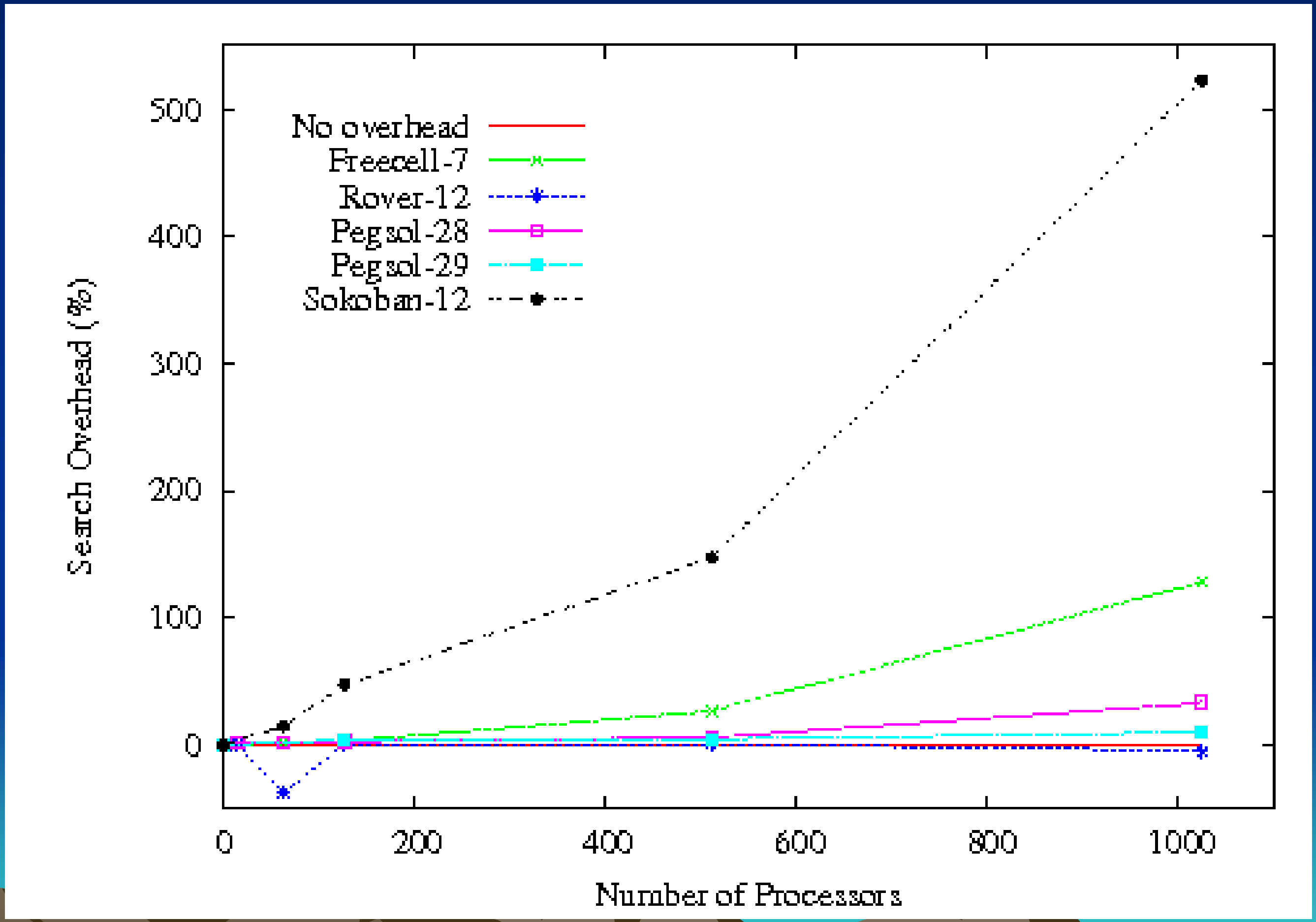
Experimental Results

- Hardware: TSUBAME
  - Each node: CPU Sun Fire X4600
    - 16 CPU cores with 32GB memory per node
  - Up to 64 nodes (= 1024 CPU cores)
- Implementation
  - Fast Downward + merge-and-shrink abstraction **[Helmert et al.:ICAPS2007]**
- Test suites
  - Problems in ICAPS Planning Competitions

Speedups in Planning

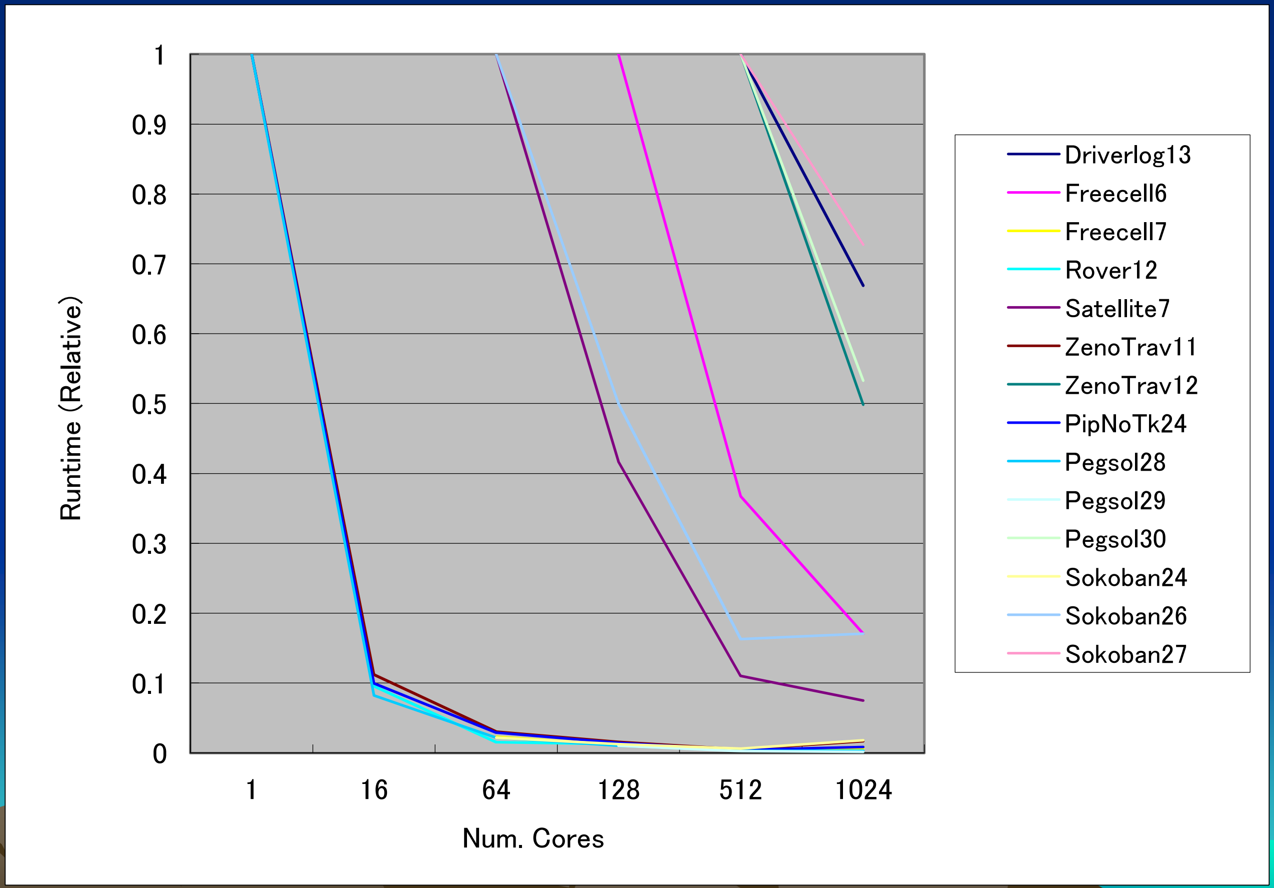


Search Overhead in Planning



Actual Efficiency of HDA* in Planning

Efficiency = *tn / tmin* ***tn*= runtime for *n* cores**
***tmin*= runtime for smallest number of cores required to solve**



Evaluating Impact of Communication Delay

- Vary # of processing nodes on a set of 64 CPU cores (4 – 64 nodes)
- 16 CPU cores per node

	1 core	**64 cores 4 nodes**	**64 cores 16 nodes**	**64 cores 64 nodes**
Satellite7	n/a	502.51	370.43	351.69
Sokoban24	2635.37	57.29	50.99	46.51

Increasing communications between processing nodes *increases* speedups

Performance Degradation Caused by Memory Contention

Normal Execution of HDA*

	1 core	64 cores 4 nodes	64 cores 16 nodes	64 cores 64 nodes
Satellite7	n/a	502.51	370.43	351.69
Sokoban24	2635.37	57.29	50.99	46.51

HDA* with dummy processes on cores unused by HDA*

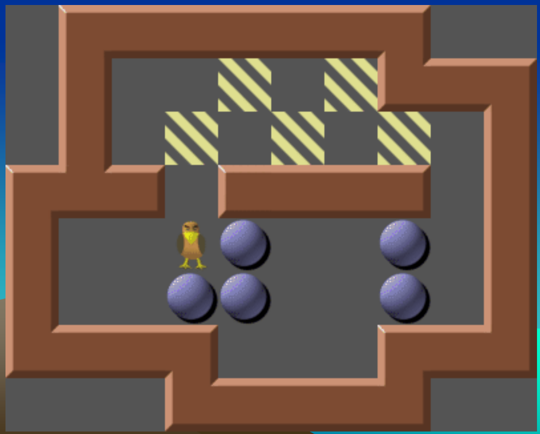
	1 core	64 cores 4 nodes	64 cores 16 nodes	64 cores 64 nodes
Satellite7	n/a	502.51	535.86	462.2
Sokoban24	2635.37	57.29	56.07	57.16

Miscellaneous

- There are problems solved only by 512 CPU cores with 1TB memory
- Load balancing *LB* ranges between 1.03 and 1.13 (128 cores)

$$LB = \frac{\textit{maximum \# of states searched by one processor}}{\textit{average \# of states searched by one processor}}$$

  – Existence of “hot spots”?



Conclusions and Future Work

- Conclusions
  - Parallel A* search applied to optimal planning
  - 55x-650x speedup on 1024 cores
- Future work
  - Invent new techniques for Grid environments
  - Utilize features of multi-core CPUs [Burns et al:2009]

Hash Distributed A* (HDA*)

- Apply TDS idea to A* search
- Prepare distributed OPEN/CLOSED lists maintained locally at each processor
- Select *S* in local OPEN
  - Check local CLOSED list for avoiding duplicate search effort
  - Generate successor *T* of S
  - Send *T* to processor that must expand
  - Routinely check if new states arrive

Other Implementation Issues

- Solution optimality
  - First solution *S* found by HDA* may not be optimal but is upper bound of optimal solution
  - Broadcast *cost*(*S*) to all processors
  - Search until best cost in OPEN >= *cost*(*S*)
- Termination detection
  - Prove that each processor has no states to expand *and no work is currently on the way*
  - Can be detected by time algorithm [Mattern:1987]