Past, Present, Parallel: Heuristic Search in a Changing Computer World

Alexander Reinefeld
Zuse Institute Berlin

16.08.2014, Prague
Disclaimer

• I’m no expert, just an observer
  – I’ve been working on other things for the last 20 years: distributed algorithms, high-performance computing systems, ...
  – Recent interest because ->

Heuristics
Algorithms
Parallelism
Heuristics
Our Contributions

• 1982: NegaScout algorithm (my MSc. thesis)

• 1992: fast interval constraint solver (with Peter Ladkin)

• 2012: MR-Search, Forward Perimeter Search (with Thorsten Schütt and Robert Döbbelin)
Trends in HPC (High-Performance Computing)

• “Computer Paradise” is just around the corner:
  – $10^x$ cores ($x \geq 7$), i.e. more than you can efficiently use
  – $10^{x-1}$ main memories
  – fast message passing

• But …
  – single cores are slow
  – no shared memory
  – resilience is an issue

S. Fuller, L. Millett, Computing Performance: Game Over or Next Level?, IEEE Computer, 1/2011, p. 33
HPC is just 5 to 10 years ahead

• Expect the power of today’s HPC on your desktop in 5 to 10 years.

• Think parallel.
Example: HPC @ Zuse Institute Berlin

1984
Cray 1M
160 MFlops

1987
Cray X-MP
471 MFlops

1994
Cray T3D
38 GFlops

1997-03
Cray T3E
486 GFlops

2002-08
IBM p690
2.5 TFlops

2008/09
SGI ICE, XE
150 TFlops

2013/14
Cray XC30
1.3 PFlops

~ 10,000,000-fold performance increase in 30 years

46,000 cores

Alexander Reinefeld
Software Challenges

• **Scalability**
  – strictly avoid sequential code, otherwise **Amdahl’s law** will hit you

\[
S_p(n) \leq \frac{1}{(1-P) + \frac{P}{N}} \leq \frac{1}{(1-P)}
\]

• **Fault-tolerance**
  – implement checkpoint/restart or redundancy

• **Balanced use of all resources**
  – CPUs, memory, interconnect
Traditional approaches don’t scale

**Search Frontier Splitting (master/slave)**
- search overhead
- communication overhead
- load imbalances

**Stack Splitting**
- good for sorted trees
- fine grained work distribution
- synchronization losses
Graphs, not Trees!

Eliminate duplicates (in parallel)

Large graphs don’t fit into memory
→ no big shared memory
→ *partition* the graph over many disjoint distributed memories and *communicate*
Optimization Possibilities

Algorithm

A*, IDA*, ...

Exploit Hardware

Parallelize, ...

Heuristic

PDBs, ...

A*,
Parallelization

• ... is tedious and error prone

• **MapReduce** [J. Dean, S. Ghemawat, OSDI 2004]
  – operates on key/value pairs
  – two functions
    • map(key, value)
    • reduce(key, value)
  – runtime system orchestrates parallel execution, data shuffling, communication, synchronization, fault tolerance, ...

• For graphs:
  – key = node
  – value = \{vertices\} or visited neighbor set
BFS with MapReduce

Process 0,

bucket of nodes

Process 1, ...

MAP
generate successors

SHUFFLE
sort nodes (global op.)

REDUCE
remove duplicates
BFS with MapReduce

void mapper(Position position, set usedMoves) {
    foreach((successor, move) \in successors(position))
        if(inverse(move) \notin usedMoves)
            emit(successor, move);
}

void reducer(Position position, set<set> usedMoves) {
    moves = \emptyset;
    foreach(move \in usedMoves)
        moves = moves \cup move;
    emit(position, moves);
}

main() {
    front = {{start_node, 0}};
    while (front \neq \emptyset) {
        intermediate = map(front, mapper());
        front = reduce(intermediate, reducer());
    }
}
Scaling BF-IDA* on a Cluster

- SGI Altix 8200 Plus
  - no globally shared memory
  - no local disk

- We used 512 nodes
  - Each node
    - 2 Intel Xeon X5570 (Nehalem EP) quad core
    - 2.93 GHz
    - 48 GB
    - 2xDDR IB network
    - separate I/O network (Lustre)
  - total: 4096 cores, 24 TB memory
Scaling BF-ID* on a Cluster

Korfs fifty 24-puzzles, hybrid MPI/OpenMP implementation on SGI

Execution Profile

Results from MR-Search, 2011
Summary: Parallelization

- MapReduce
  - works for BFS and BF-IDA*
  - scales well to several thousand CPUs
    - but has a few subtlenesses
  - works for clusters and SMPs
Optimization Possibilities

Algorithm

A*, IDA*, ...

Exploit Hardware

Heuristic

Parallelize, ...

PDBs, ...
Heuristics

• abstract from the original problem; solve simpler problem
  – Manhattan distance
  – Pattern databases (PDBs)
  – ...

• need *admissible* heuristics
PDBs

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

run parallel BFS and count moves

Lookup-Table: 9.5 TB
perfect heuristic

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

run parallel BFS and count moves

Lookup-Table: 5.5 MB

+ lookup table
- poor heuristic
- only max-of
don’t care tiles
## Additive PDBs

1. Divide into disjoint tile sets
2. Run 3x BFS
3. Sum over all three table lookups

### Problems:
- Not allowed to count moves of don’t cares
- Many possible partitions. Finding a good one is “art”.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
Additive PDBs (II)

1. Build pattern space
   - merge node from the puzzle graph, if pattern tiles in the same position*)

2. Run BFS over pattern space and count moves
   - compute min

*) Two states of the original space map to the same state in the pattern space, if the pattern tiles are in the same position and the two blank positions can be reached from each other by moving only don’t care tiles.
### Size of PDBs for 24-puzzle

<table>
<thead>
<tr>
<th>#Tiles</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>121 MB</td>
</tr>
<tr>
<td>7</td>
<td>2 GB</td>
</tr>
<tr>
<td>8</td>
<td>40 GB</td>
</tr>
<tr>
<td>9</td>
<td>690 GB</td>
</tr>
<tr>
<td>10</td>
<td>10 TB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PDB</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-6-6-6</td>
<td>488 MB</td>
</tr>
<tr>
<td>8-8-8</td>
<td>122 GB</td>
</tr>
<tr>
<td>9-8-7</td>
<td>733 GB</td>
</tr>
<tr>
<td>9-9-6</td>
<td>1381 GB</td>
</tr>
</tbody>
</table>
Evaluating PDBs

- **SGI UV**
  - 512 cores
  - 2 TB shared memory (NUMA)

- parallel file-system

9-9-6 = 1.381 TB, rest for search
Evaluating PDBs (II)

1. Load PDBs into main memory (slow)
2. Run BF-IDA* on 50 puzzles

• max-of
  – 6-6-6-6
  – 8-8-8
  – 9-8-7
  – 9-9-6

Note: PDBs b, c, d partly overlap, saves space
Evaluating PDBs (fifty 24 puzzles)

sorted by node expansions with 6-6-6-6 PDB

0.71 (slowdown)
Evaluating PDBs (fifty 24 puzzles)
Cumulative Distribution

100,000,000 random instances
Summary: PDBs

• large PDBs can be built (BF MapReduce)
• 8x reduction by 9-9-6 vs. 6-6-6-6 PDB
• Many outliers $\rightarrow$ max-of is better (9.36x)
• Did we create a good 9-9-6?

Optimization Possibilities

Algorithm

A*, IDA*, ...

Exploit Hardware

Parallelize, ...

Heuristic

PDBs, ...

A*,
Iterative Deepening A*

```cpp
int thresh = h(start);
bool found = false;
while(!found) {
    found = test(start, goal, h, thresh);
    increase thresh
}

**test** – Is there a path of length thresh from start to goal?
**test-df** – depth-first search with heuristic pruning
**test-bf** – breadth-first search with heuristic pruning
test: DF vs. BF

test-df

test-bf
test: DF vs. BF (fail-case)

BF $\geq$ DF

BF is optimal, DF depends on transpositions
test: DF vs. BF (success-case)

BF ≠ DF

• DF: may be lucky in the last iteration, but it cannot outperform BF in previous iterations
Wish-list for test

- use BF for failure-case

- learn tree characteristics from prev. iterations
Algorithm: Forward Perimeter Search

foreach p in perimeter
    test-bf(p, goal, h, thresh-r)

+ smaller trees
- trees overlap
Algorithm: Forward Perimeter Search

\[
\text{foreach } p \text{ in } \text{sort}(\text{perimeter}) \\
\text{test-bf}(p, \text{goal, } h, \text{thresh-r})
\]
Algorithm: Forward Perimeter Search

```
expand_perimeter();
foreach p in sort(perimeter)
    test-bf(p, goal, h, thresh-d(start,p))
```

- trees overlap
+ early hit in last iter
+ can control tree size

→ available memory
→ reduce overlap
Test Algorithm

test-fps() {
    foreach p in perimeter
        if(p.tree > limit)  // size or width
            expand_perimeter(perimeter, start, p, r)
    foreach p in sort(perimeter)
        if(test(p, goal, h, thresh-d(start,p)))
            return true;  // done
    return false;
}

expand_perimeter(P, start, p, r) {
    P -= {p};
    C = circle(p, r);
    foreach n in C
        if(d(start,n) == d(start,p)+r)
            P += n;  // triangular eq.
    return P
}
# Evaluation

(hardest 24-puzzle from [Korf], 113 moves to solve, 6-6-6-6 PDB)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Node Expansions</th>
<th>Memory [Nodes]</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDA*</td>
<td>4,156,099,168,506</td>
<td>113</td>
</tr>
<tr>
<td>BF-IDA*</td>
<td>1,067,321,687,213</td>
<td>50,675,640,000</td>
</tr>
<tr>
<td>FPS, r=4</td>
<td>423,306,411,815</td>
<td>5,922,529,960</td>
</tr>
<tr>
<td>FPS, r=6</td>
<td>428,072,054,940</td>
<td>2,876,547,362</td>
</tr>
<tr>
<td>FPS, r=10</td>
<td>564,996,269,605</td>
<td>1,220,873,196</td>
</tr>
<tr>
<td>FPS, r=16</td>
<td>647,863,040,082</td>
<td>503,869,879</td>
</tr>
<tr>
<td>FPS, r=18</td>
<td>671,310,216,245</td>
<td>257,590,848</td>
</tr>
<tr>
<td>FPS, 1.88 \cdot 10^8</td>
<td>404,811,541,671</td>
<td>1,437,995,218</td>
</tr>
<tr>
<td>FPS, 3.5 \cdot 10^7</td>
<td>452,935,148,947</td>
<td>1,078,733,091</td>
</tr>
<tr>
<td>FPS, 1.7 \cdot 10^7</td>
<td>486,941,686,873</td>
<td>457,659,207</td>
</tr>
<tr>
<td>FPS, 2 \cdot 10^6</td>
<td>619,262,051,017</td>
<td>112,469,403</td>
</tr>
<tr>
<td>FPS, 1 \cdot 10^6</td>
<td>652,659,857,757</td>
<td>54,618,898</td>
</tr>
</tbody>
</table>

* IDA* halts at 1st solution, whereas BF-IDA* returns all solutions

- runs on PC
- 256 node cluster
- PC (8 GB)
Effect of large perimeters
Summary: FPS

- expands fewer nodes than IDA* and BF-IDA*
- is highly parallel
- allows to control memory consumption

Conclusion: How to solve really large problems?

• Utilize *all* available resources: CPUs, memories, and interconnect in a balanced way.

• Build some good PDBs with parallel BFS
  – on SMP or clusters
  – *Which info is most beneficial?*

• Search the solution space with FPS
  – highly parallel
  – flexible use of memory
Final word

“He who comes too late is punished by life.”

Mikhail Gorbachev, East-Berlin, 7 October 1989