Playing Zero-Sum Games on the GPU

Avi Bleiweiss
NVIDIA Corporation

San Jose | 2010
Zero-Sum

- One’s gain, other’s loss
- Perfect info
- Multi player game
  - Simple and involved

<table>
<thead>
<tr>
<th>Matching Pennies</th>
<th>Head</th>
<th>Tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>1,-1</td>
<td>-1,1</td>
</tr>
<tr>
<td>Tail</td>
<td>-1,1</td>
<td>1,-1</td>
</tr>
</tbody>
</table>
Deep Blue

- 32 CPUs
- 512 Chess ASICs
- 1.15B transistors

Graph showing Million Moves/Seconds from 1985 to 1997:
- 1985: 0
- 1988: 50
- 1991: 100
- 1994: 150
- 1997: 200

Attributes of Deep Blue:
- 6’5” tower pair
- 1.4 tons
- 100M$
Motivation

- Play as you go
  - Thousands players
- Game cloud
  - GPU computing
- Mobile computer
  - 3D graphics

NVIDIA Tesla

NVIDIA Tegra
Problem

Game
- Two player
- Maximize look ahead
- Complexity
  - $10^{25}$ for 4x4x4 Tic-Tac-Toe
  - $10^{120}$ for Chess

Search
- Efficient parallel
  - Single game
  - Simultaneous matches
- Graceful stack
- Linear scale
Game Tree
Mini-Max

- Build and search
- Unbound DFS
  - All nodes visited
- Non tail recursive
- Depth limited

```java
MinMax(node, max)
if(isLeaf(node))
    return evaluate(node)
if(max) then rank ← −∞
else rank ← ∞
foreach(s_i in successors(node)) do
tmp ← MiniMax(s_i, !max)
if(max) then
    rank ← MAX(rank, tmp)
else
    rank ← MIN(rank, tmp)
return rank
```
Principal Variation
Alpha-Beta

- Enhances Mini-Max
- Elegant, efficient
  - Prunes nodes
- Perfect game
  - Possible in cases

```
AlphaBeta(node, max, α, β)
// termination, initialization
if (α > β) return rank

foreach(s_i in successors(node)) do
    tmp ← AlphaBeta(s_i, !max, α, β)
    if (max) then
        rank ← MAX(rank, tmp)
        α ← MAX(α, rank)
    else
        rank ← MIN(rank, tmp)
        β ← MIN(β, rank)

return rank
```
Pruning

![Diagram of a pruned tree with nodes labeled A, B, C, D, E, F, G, H. The tree has a root node and branches with values such as 7, 4, and -∞. The diagram includes arrows indicating the direction of the tree structure.]
Optimization

Aspiration Search
- Tree value known
- Narrow window
- Re-search if fails

Principal Variation Search
- Non PV nodes
  - \( \beta = \alpha + 1 \)
  - Full window PV nodes
  - Perfect ordered tree

Iterative Deepening
- Fixed depth searches
- Transposition table
- Hash branch positions

[Marsland, T. A. 1986]
Parallelism

Principal Variation Split
- Strongly ordered tree
- Synchronization bound
- Load imbalance

Young Brothers Wait Concept
- Parallel at any node
- Processor owns node
- Scales to # processors

Dynamic Tree Splitting
- Processors share node
- Global job list
- Reasonable speedup

[Feldmann, R. 1993]
Challenges

- Deep recursion, limited stack
- Divergent, irregular threads
- Dynamic parallelism
- Low arithmetic intensity
Implementation

- Kernel for each
  - Mini-Max, Alpha-Beta
- Board C++ class
  - Rules specific
- Games
  - 3D Tic-Tac-Toe
  - Connect-4
  - Reversi
Board

Pitched 2D/3D

- Cells
- Successors
  - Player
  - Move
  - struct Move {
    int3 position;
    int rank;
  };

Player
{none, 1st, 2nd}

Rank
{Win, Lose, Draw}

Manipulate
- Update
- Undo
- Winner
- Full

Query

Winner
Full
Stack

- Recursion depth >1000
- Greedy allocation

\[ \text{StackBytes/Thread} \times \text{SM} \# \times \text{MaxWarps/SM} \times \text{Threads/Warp} \]

- Hybrid design

  - Local Memory
    - Local variables
    - Function parameters

  - Runtime/Compiler

  - User
    - Successors

  - Global Memory
Split

Find highest cut nodes

Parallel node search

Resolve up to root

Thousands of working threads

Game Tree

foreach move

game init

game over

producer
consumer

CPU
GPU

foreach move
Shared αβ

```c
__device__ int galpha, gbeta;

__device__ void resolve(int alpha, int beta)
{
    if (alpha <= galpha) alpha = galpha;
    else atomicMax(&galpha, alpha);

    if (beta >= gbeta) beta = gbeta;
    else atomicMin(&gbeta, beta);
}
```
Limitations

- Stack allocation
  - Bounds parallelism
- Split constraints

<table>
<thead>
<tr>
<th>Depth</th>
<th>Tic-Tac-Toe Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3x3x3</td>
</tr>
<tr>
<td>1</td>
<td>650</td>
</tr>
<tr>
<td>2</td>
<td>15600</td>
</tr>
</tbody>
</table>
Methodology

- CUDA Toolkit 3.1, Windows
- Processors

<table>
<thead>
<tr>
<th>GPU</th>
<th>SMs</th>
<th>Warps/SM</th>
<th>Clocks (MHz)</th>
<th>L1/Shared (KB)</th>
<th>L2 (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX480</td>
<td>15</td>
<td>2</td>
<td>723/1446/1796</td>
<td>48/16</td>
<td>640</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU</th>
<th>Cores</th>
<th>Clocks (MHz)</th>
<th>L1/L2 (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I7-940</td>
<td>8</td>
<td>2942/(3*1066)</td>
<td>32/8192</td>
</tr>
</tbody>
</table>
Single Game

4x4x4 Tic-Tac-Toe

- Naïve Split
- Shared αβ

Lower is good
Simultaneous Matches

4x4x4 Tic-Tac-Toe

Average Seconds/Move

Matches

GTX480  i7 8 Threads

lower is good

higher is good

Speedup
Game Analysis

4x4x4 Tic-Tac-Toe

- GTX480
- i7 8 Threads

- 4K Matches

Seconds per Move vs. Move #

Lower is better
Future Work

- Data packing
- Backtracking search
- Sudoku, toy game
  - Generator, solver
- Multi recursion
## GPU Performance

<table>
<thead>
<tr>
<th>Metric</th>
<th>Game</th>
<th>Dimension</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared αβ vs. Naïve Split</td>
<td>Tic-Tac-Toe</td>
<td>4x4x4</td>
<td>13.37X mean</td>
</tr>
<tr>
<td>Simultaneous Matches vs. CPU</td>
<td>Tic-Tac-Toe</td>
<td>4x4x4</td>
<td>5.22X @ 16K</td>
</tr>
<tr>
<td></td>
<td>Connect-4</td>
<td>7x6</td>
<td>6.26X @ 32K</td>
</tr>
<tr>
<td></td>
<td>Reversi</td>
<td>8x8</td>
<td>5.96X @ 16K</td>
</tr>
</tbody>
</table>
Summary

- Efficient hybrid stack
- Dynamic parallelism
- Tegra, Tesla solution
- 3D Chess on GPU!

<table>
<thead>
<tr>
<th></th>
<th>Deep Blue</th>
<th>Tesla</th>
</tr>
</thead>
<tbody>
<tr>
<td>$/\text{(Moves}/\text{Sec})$</td>
<td>0.5</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Courtesy freegames4all
Thank You!

Questions?
Info

- **Base**

- **GPU AI**
  - [Technology Preview](http://developer.nvidia.com)

- **Toolkit**
  - [CUDA Zone](http://developer.nvidia.com)

- **Debugger**
  - [Parallel Nsight](http://developer.nvidia.com)
Backup
Simultaneous Matches (1)

![Graph showing average seconds per move for 7x6 Connect-4 with GTX480 and i7 8 Threads. The x-axis represents the number of matches, and the y-axis represents the average seconds per move. The graph compares the performance of GTX480 and i7 8 Threads, with lower values indicating better performance. The x-axis labels are 1, 16, 128, 1024, 4096, 16384, and 32768 matches. The y-axis labels are 0, 0.5, 1, 1.5, 2, 2.5. The graph includes notes indicating that lower is good and higher is good. The graph is presented by NVIDIA.](image)
Simultaneous Matches (2)

8x8 Reversi

Average Seconds / Move

GTX480  i7 8 Threads

lower is better

higher is better
Simultaneous Puzzles

9x9 Sudoku

Seconds

Puzzles

Speedup

lower is better

higher is better

GTX480

i7 8 Threads