UTS: An Unbalanced Tree Search Benchmark

LCPC 2006
Coauthors

- Stephen Olivier, UNC
- Jun Huan, UNC/Kansas
- Jinze Liu, UNC
- Jan Prins, UNC
- James Dinan, OSU
- P. Sadayappan, OSU
- Chau-Wen Tseng, UMD

- Also, thanks to Ohio Supercomputer Center, European Center for Parallelism in Barcelona, and William Pugh
Motivating the Problem

• Suppose we have a large search space and a tree-structured search.
• How could we do this in parallel?
Motivating the Problem

- Suppose we have a large search space and a tree-structured search.
- How could we do this in parallel?
- Now suppose the tree is highly unbalanced.
Meet UTS

- Unbalanced computation requiring continuous dynamic load balancing for high performance

- Exposes trade-offs between communication costs and granularity of load balance
  - Fundamentally disadvantages systems with coarse-grained communication networks

- Challenges the expressibility and performance portability of parallel programming languages
Benchmark Characteristics

- Implicit Generation
  - Tree nodes generated on the fly
    - Mimics systematic search
    - Well-suited for parallel exploration

- Imbalance
  - Large variation in subtree sizes

- Parameterization
  - Can create trees of different depths and sizes
Outline

• Tree Generation

• Implementation

• Performance Analysis

• Conclusions and Future Work
Generation Strategy

- Each node consists of a 20B descriptor
- Implicit child creation using cryptographic hash

\( (\text{Parent Descriptor, Child Index}) \)

\( \text{Hash} \)

\( \text{Child Node Descriptor} \)

- Number of children determined by sampling a probability distribution using the descriptor
Generating Binomial Trees

- In a Binomial Tree, each node has 0 or \( m \) children
- A node has children with probability \( q \)
- When \( qm < 1 \), expected size is \( \frac{1}{1 - qm} \)
- As \( qm \) approaches 1, subtree size variation increases (**Imbalance!**)
- Expected subtree size rooted at each node is independent of its location in the tree

How large a subtree at each?

...
Generating Binomial Trees

- In a Binomial Tree, each node has 0 or \( m \) children
- A node has children with probability \( q \)
- When \( qm < 1 \), expected size is \( \frac{1}{1 - qm} \)
- As \( qm \) approaches 1, subtree size variation increases (Imbalance!)
- Expected subtree size rooted at each node is independent of its location in the tree

How large a subtree at each? 
... Don’t know until we’ve explored them ...

\[
1 - qm
\]
Generating Binomial Trees

• In a Binomial Tree, each node has 0 or \( m \) children
• A node has children with probability \( q \)
• When \( qm < 1 \), expected size is \( \frac{1}{1 - qm} \)
• As \( qm \) approaches 1, subtree size variation increases (Imbalance!)
• Expected subtree size rooted at each node is independent of its location in the tree

Now we know
Generating Geometric Trees

- Use geometric distribution to determine the number of children
- Expected value specified as a parameter
- Depth limitation without which trees could grow indefinitely
- Long tail of geometric distribution can cause some nodes to have many more children than others (Imbalance!)
Generating Geometric Trees

- Use geometric distribution to determine the number of children
- Expected value specified as a parameter
- Depth limitation without which trees could grow indefinitely
- Long tail of geometric distribution can cause some nodes to have many more children than others (Imbalance!)

“Lucky” sampled value from tail
Implementation

• Goal is to explore the entire tree
• Each thread performs depth first traversal starting from a given node using a stack
• When a thread runs out of work, it must be able to acquire more nodes
  ➢ Work sharing - busy threads with surplus work responsible for offloading to idle threads
  ➢ Work stealing - idle threads responsible for seeking out surplus work at other threads
    ❖ Busy threads always making progress on goal
Implementation Details

- How to express in OpenMP & UPC?
  - Shared variables to express program and thread state
  - Concurrency control only when necessary due to high expense
- Local stack operations performed without locking
  - Most frequent operations
- Steal operations performed on shared stack with locking

![Stack Diagram]

- Stack Top
  - Local Access Only
  - Shared Access
  - Steal Stack

*Figure 2. A thread's steal stack tree type*
Implementation Details

- How to express in OpenMP & UPC?
  - Shared variables to express program and thread state
  - Concurrency control only when necessary due to high expense
- Local stack operations performed without locking
  - Most frequent operations
- Steal operations performed on shared stack with locking

![Stack Diagram]

No Locking Here

Stack Top

Local Access Only

Shared Access

Steal Stack

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL
Implementation Details

• How to express in OpenMP & UPC?
  ➢ Shared variables to express program and thread state
  ➢ Concurrency control only when necessary due to high expense

• Local stack operations performed without locking
  ➢ Most frequent operations

• Steal operations performed on shared stack with locking
Sample Trees

<table>
<thead>
<tr>
<th>Tree</th>
<th>Type</th>
<th>$b_0$</th>
<th>$d$</th>
<th>$q$</th>
<th>$m$</th>
<th>$r$</th>
<th>Depth</th>
<th>MNodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Geometric</td>
<td>4</td>
<td>10</td>
<td>–</td>
<td>–</td>
<td>19</td>
<td>10</td>
<td>4.130</td>
</tr>
<tr>
<td>T2</td>
<td>Geometric</td>
<td>1.014</td>
<td>508</td>
<td>–</td>
<td>–</td>
<td>0</td>
<td>508</td>
<td>4.120</td>
</tr>
<tr>
<td>T3</td>
<td>Binomial</td>
<td>2000</td>
<td>–</td>
<td>0.124875</td>
<td>8</td>
<td>42</td>
<td>1572</td>
<td>4.113</td>
</tr>
</tbody>
</table>

- All have nearly the same number of nodes but different depths
- $r$ is seed for the root node, and can be varied to generate different trees in the same parameter class
Sample Trees

<table>
<thead>
<tr>
<th>Tree</th>
<th>Type</th>
<th>$b_0$</th>
<th>$d$</th>
<th>$q$</th>
<th>$m$</th>
<th>$r$</th>
<th>Depth</th>
<th>MNodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Geometric</td>
<td>4</td>
<td>10</td>
<td>–</td>
<td>–</td>
<td>19</td>
<td>10</td>
<td>4.130</td>
</tr>
<tr>
<td>T2</td>
<td>Geometric</td>
<td>1.014</td>
<td>508</td>
<td>–</td>
<td>–</td>
<td>0</td>
<td>508</td>
<td>4.120</td>
</tr>
<tr>
<td>T3</td>
<td>Binomial</td>
<td>2000</td>
<td>–</td>
<td>0.124875</td>
<td>8</td>
<td>42</td>
<td>1572</td>
<td>4.113</td>
</tr>
</tbody>
</table>

- All have nearly the same number of nodes but different depths
- $r$ is seed for the root node, and can be varied to allow different trees with other parameters the same
Sample Trees

<table>
<thead>
<tr>
<th>Tree</th>
<th>Type</th>
<th>$b_0$</th>
<th>$d$</th>
<th>$q$</th>
<th>$m$</th>
<th>$r$</th>
<th>Depth</th>
<th>MNodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Geometric</td>
<td>4</td>
<td>10</td>
<td>–</td>
<td>–</td>
<td>19</td>
<td>10</td>
<td>4.130</td>
</tr>
<tr>
<td>T2</td>
<td>Geometric</td>
<td>1.014</td>
<td>508</td>
<td>–</td>
<td>–</td>
<td>0</td>
<td>508</td>
<td>4.120</td>
</tr>
<tr>
<td>T3</td>
<td>Binomial</td>
<td>2000</td>
<td>–</td>
<td>0.124875</td>
<td>8</td>
<td>42</td>
<td>1572</td>
<td>4.113</td>
</tr>
</tbody>
</table>

- All have nearly the same number of nodes but different depths
- $r$ is seed for the root node, and can be varied to allow different trees with other parameters the same
### Sequential Exploration Rates

<table>
<thead>
<tr>
<th>System</th>
<th>Processor</th>
<th>Compiler</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray X1</td>
<td>Vector (800 MHz)</td>
<td>Cray 5.5.0.4</td>
<td>29</td>
<td>29</td>
<td>31</td>
</tr>
<tr>
<td>SGI Origin 2000</td>
<td>MIPS (300 MHz)</td>
<td>GCC 3.2.2</td>
<td>158</td>
<td>156</td>
<td>170</td>
</tr>
<tr>
<td>SGI Origin 2000</td>
<td>MIPS (300 MHz)</td>
<td>Mipspro 7.3</td>
<td>169</td>
<td>166</td>
<td>183</td>
</tr>
<tr>
<td>Sun SunFire 6800</td>
<td>Sparc9 (750 MHz)</td>
<td>Sun C 5.5</td>
<td>260</td>
<td>165</td>
<td>384</td>
</tr>
<tr>
<td>P4 Xeon Cluster (OSC)</td>
<td>P4 Xeon (2.4GHz)</td>
<td>Intel 8.0</td>
<td>717</td>
<td>940</td>
<td>1354</td>
</tr>
<tr>
<td>Mac Powerbook</td>
<td>PPC G4 (1.33GHz)</td>
<td>GCC 4.0</td>
<td>774</td>
<td>672</td>
<td>1117</td>
</tr>
<tr>
<td>SGI Altix 3800</td>
<td>Itanium2 (1.6GHz)</td>
<td>GCC 3.4.4</td>
<td>951</td>
<td>902</td>
<td>1171</td>
</tr>
<tr>
<td>SGI Altix 3800</td>
<td>Itanium2 (1.6GHz)</td>
<td>Intel 8.1</td>
<td>1160</td>
<td>1106</td>
<td>1477</td>
</tr>
<tr>
<td>Dell Blade Cluster (UNC)</td>
<td>P4 Xeon (3.6GHz)</td>
<td>Intel 8.0</td>
<td>1273</td>
<td>1165</td>
<td>1866</td>
</tr>
</tbody>
</table>

- **Performance given in kNodes/sec**
- **Note the variation with tree type (T1 and T2 are geometric trees; T3 is a binomial tree)**

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL
Performance on Shared Memory (SGI Origin 2000)

Fig. 3. Parallel performance on the Origin 2000 using OpenMP and UPC. On the left, results for geometric trees at UNCs. The cluster's interconnect is Infinib and we configured the Berkeley 8pe. Txqrtation Perf attri for UPC and base on UPC of CL and the version. The tree's inelcations when only 8e and dynamic tree will be explored. The cluster's interconnect is Infinib and we configured the Berkeley 8pe. Txqrtation Perf attri for UPC and base on UPC of CL and the version. The tree's inelcations when only 8e and dynamic tree will be explored.

The UNIVERSITy of NORTH CAROLINA at CHAPEL HILL
Performance on Shared Memory (SGI Origin 2000)

Geometric Trees (T1, T2)

Performance (Millions of nodes / sec)

Number of Processors

T2 - OpenMP
T1 - OpenMP
T2 - UPC
T1 - UPC

UPC Slower
Performance on Shared Memory (SGI Origin 2000)

Geometric Trees (T1, T2)

Performance (Millions of nodes / sec)

Number of Processors

T1 vs T2
Performance on Distributed Memory

Sample Trees on the Dell Cluster using UPC

Performance (Millions of nodes per second) vs. Number of Processors

- T3
- T2
- T1

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL
Granularity and Scaling

- Geometric tree T1 on the SGI Origin 2000
- Chunk size governs the amount of locking and communication
Granularity and Scaling

- Geometric tree T1 on the SGI Origin 2000
- Chunk size governs the amount of locking and communication

Stealing overhead dominates
Granularity and Scaling

- Geometric tree T1 on the SGI Origin 2000
- Chunk size governs the amount of locking and communication
Granularity and Scaling

- Geometric tree T1 on the SGI Origin 2000
- Chunk size governs the amount of locking and communication
Comparing Trees & Systems While Varying Granularity

T1 (Geometric)

T3 (Binomial)

Origin Performance (Mnodes / sec)

Cluster Performance (Mnodes / sec)

Chunk size (linear scale)

Chunk size (log scale)
Comparing Trees & Systems
While Varying Granularity

T1 (Geometric)

Origin Performance (Mnodes/sec)

Cluster Performance (Mnodes/sec)

Chunk size (linear scale)

T3 (Binomial)

Chunk size (log scale)!
Comparing Trees & Systems While Varying Granularity

T1 (Geometric)  T3 (Binomial)

Origin Performance (Mnodes / sec)

Cluster Performance (Mnodes / sec)

Sensitive to chunk size

Chunk size (linear scale)  Chunk size (log scale!)
Comparing Parallel Efficiency

- Efficiency at optimal chunk size for each machine
- Near-ideal scaling on the Origin
  - UPC and OpenMP
- Poor scaling on the cluster
Comparing Parallel Efficiency

- Efficiency at optimal chunk size for each machine
- Near-ideal scaling on the Origin
  - UPC and OpenMP
- Poor scaling on the cluster
Comparing Thread Behavior

- Dark blue is work time
- Light blue is time searching for work
- White is idle time
Comparing Thread Behavior

- Dark blue is work time
- Light blue is time searching for work
- White is idle time

Origin 2000 (16 threads)

Cluster (4 & 16 threads)

Trouble finding work
Comparing Thread Behavior

- Dark blue is work time
- Light blue is time searching for work
- White is idle time

Origin 2000 (16 threads)

Cluster (4 & 16 threads)
Expressing Locality OpenMP vs. UPC on Shared Memory

- OpenMP lacks portable explicit locality specifier
- Results in false sharing in on some systems
- Problem resolved using dynamic allocation
- Results shown for Origin 2000
Expressing Locality OpenMP vs. UPC on Shared Memory

- OpenMP lacks portable explicit locality specifier
- Results in false sharing in on some systems
- Problem resolved using dynamic allocation
- Results shown for Origin 2000
Expressing Locality OpenMP vs. UPC on Shared Memory

- OpenMP lacks portable explicit locality specifier
- Results in false sharing in on some systems
- Problem resolved using dynamic allocation
- Results shown for Origin 2000
Expressing Locality OpenMP vs. UPC on Shared Memory

No problems for UPC either way

- OpenMP lacks portable explicit locality specifier
- Results in false sharing in on some systems
- Problem resolved using dynamic allocation
- Results shown for Origin 2000
Larger Problem (Tree) Size

- Altix 3800 -- 64 processors used
- Both trees are same type (binomial)
Larger Problem (Tree) Size

- Altix 3800 -- 64 processors used
- Both trees are same type (binomial)
Comparing Performance on Various Machines

- T3 using UPC on 16 processors
- If it scaled better, Blade cluster would outperform the Altix 3800
Issues Addressed

• Measure the ability of parallel systems to perform continuous dynamic load balancing

• Understand the trade-offs between communication costs and granularity of load balance
  ➢ On shared memory and on distributed memory

• Compare parallel programming languages
  ➢ Expressibility and ease of programming
  ➢ Portability and performance portability
Conclusions

- **UPC versus OpenMP on shared memory**
  - Similar in ease of programming
  - UPC has superior facility for specifying locality
  - Both exhibit near-ideal scaling
  - Comparable absolute performance

- **UPC implementation on distributed memory**
  - Frequent access of shared variables for coordinating the work stealing and termination detection lead to poor performance and scaling

- **Several factors impact performance**
  - Data locality
  - Granularity of load balance
  - Problem size
Ongoing and Future Work

• MPI versions of UTS
  ➢ Work sharing and work stealing methods

• Further tuning of UPC version, especially to improve distributed memory performance
  ➢ Reduce/replace many shared variable accesses
    ❖ Lose advantage of UPC abstractions for programmability
    ➢ Trade-off with possible decrease in shared memory performance
      ❖ Fails to provide performance portability

• Performance testing on other systems