Improving Scalability of OpenMP Applications on Multi-core Systems Using Large Page Support

Ranjit Noronha and Dhabaleswar K. Panda
Network Based Computing Laboratory (NBCL)
The Ohio State University
Outline of the talk

- Introduction and Motivation
- Potential Issues
- Experimental Evaluation
- Conclusions and Future Work
Page Table Architecture

- Page walk may take up to two memory accesses
- May be a substantial overhead
- TLB misses may be expensive
Processor Large Page Support

• Traditional page size is 4KB
• Other page sizes
  – 2MB, 4MB supported by Intel and AMD processors
  – Itanium supports a variety of sizes
• Benefits of large pages
  – Fewer TLB misses
  – Larger memory coverage
• Drawbacks of large pages
  – Fewer TLB entries
  – Page misses are more expensive
• TLB Entries for 4KB pages and 2MB are generally separate
  – Eg. Xeon has 128 entries for 4KB pages and 32 entries for 2MB pages
• Memory region under TLB coverage is affected
• For contiguous regions, large pages might provide benefit
• For non-contiguous regions, there might be increased TLB misses for large pages
Introduction to OpenMP

- OpenMP is a language specification
  - Annotation of sequential C, C++, and Fortran programs. Annotations produce:
    - Creation of thread teams
    - Execution in parallel

- Primitives for synchronization
  - Locks
  - Critical sections
  - Single threaded regions

- Primitives for variable sharing
  - Private or Shared
  - Variable initialization for Parallel Regions
OpenMP and Large Pages

- Loop Level Parallelism
  - May use an array as a data structure
  - Large arrays may potentially span several 4KB pages
  - Each thread may work on a different portion of the array
  - More complicated access patterns
    - Strided access such as FFT
    - Array of structures
  - Access locality
    - Threads might experience several TLB misses
    - TLB misses are expensive
- DTLB and ITLB are shared in SMT’s
  - But large pages have fewer TLB entries
Motivation

- Potential Strategies for providing large pages to memory allocations to OpenMP applications
- Are static or dynamic page allocations strategies more appropriate
- Are large pages beneficial for instruction footprints
- Are large pages beneficial for application data footprint
- What is the impact of large pages on application scalability on different processor architectures
- Is there an impact of large pages when the TLB is shared between hyperthreads
Potential Strategies for providing large pages

• OpenMP applications
  – Assumption of shared memory
  – Stack and heap variables allocations shared on a node
  – Stack and heap variables should use large pages
• Using large pages for stack variables
  – Need to modify the compiler
• Employing large pages for heap variables
  – Modify libc
• Deploy a memory mapped file
  – Allocate space for stack and heap variables from the memory mapped file
  – Translate stack allocations to heap allocations
  – Omni/SCASH Cluster OpenMP performs this translation
  – Disable all Cluster OpenMP coherency features
  – Memory mapped file uses *hugetlbfs* for large pages
Page Allocation Strategies

• Studies for large page allocation
  – On demand
  – Contention from many processes

• Characteristics of OpenMP applications
  – Parallel codes
  – Likely to be only application on the node
  – Some OpenMP applications use only stack variables (eg. NAS)
  – Static reservation of a pool of large pages likely an acceptable tradeoff
  – Reduce complexity and latency of memory allocation
Outline of the talk

• Introduction and Motivation
• Potential Issues
• Experimental Evaluation
• Conclusions and Future Work
Experimental Setup

• Platform 1
  – Dual dual-core (4-cores) Opteron 270 processors
  – 4GB memory
  – 2.0 GHz

• Platform 2
  – Dual dual-core (4-cores) Xeon based platform
  – Hyperthreading enabled (2threads/core or 8 threads totally)
  – 12GB memory
  – 3.2 GHz

• Applications
  – NAS OpenMP codes CG, SP, MG, FT and BT
  – Class B
## Application Memory Footprint

<table>
<thead>
<tr>
<th></th>
<th>Instruction</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT (B)</td>
<td>1.6MB</td>
<td>371MB</td>
</tr>
<tr>
<td>CG (B)</td>
<td>1.4MB</td>
<td>725MB</td>
</tr>
<tr>
<td>FT (B)</td>
<td>1.4MB</td>
<td>2.4GB</td>
</tr>
<tr>
<td>SP (B)</td>
<td>1.6MB</td>
<td>387MB</td>
</tr>
<tr>
<td>MG (B)</td>
<td>1.4MB</td>
<td>884MB</td>
</tr>
</tbody>
</table>

- Instruction binary footprint are all less than 2MB
- Data footprint much larger
- ITLB coverage for instruction data is large with 4KB pages
- Instruction locality high (most time spent in parallel loops)
Instruction TLB Misses

- ITLB misses/second of application time under 0.45 misses/sec
- Not likely to a substantial source of overhead
- We do not use large pages for instruction binaries
System Scalability (CG)

- 25% improvement in performance at 4 threads (Opteron)
System Scalability (SP)

- 20% improvement in performance at 4 threads on Opteron
System Scalability (MG)

- 17% improvement in performance at four threads on Opteron
System Scalability (FT, BT)

- No significant impact from large pages
- Mainly because of the data distribution
Data TLB Misses

![Bar chart showing Data TLB Misses for different applications and two different cache sizes (4KB and 2MB).]
Conclusions and Future Work

- Explored using large pages for OpenMP codes
- 25% improvement at 4 threads on Opteron
- Large Pages Improve Scalability of some NAS OpenMP applications
- Contention and Memory Bandwidth Limit with hyperthreading limit improvement on the Intel Xeons
- Explore using a mix of small and large pages
- Use large pages for Cluster OpenMP applications
Acknowledgements

Our research is supported by the following organizations

• Current Funding support by
  - Office of Science
  - NSF
  - Mellanox Technologies
  - Cisco Systems
  - Intel
  - Linux Networx
  - NetApp

• Current Equipment support by
  - Mellanox Technologies
  - Intel
  - AMD
  - Sun Microsystems
  - APPRO
  - HPC Cluster Solutions
  - AMMASS
  - IBM
  - Microway
  - PathScale
  - SilverStorm
Web Pointers

http://nowlab.cse.ohio-state.edu/
noronha@cse.ohio-state.edu
Backup Slides
Introduction to OpenMP

- OpenMP is a language specification
  - Annotation of sequential C, C++, and Fortran programs.
    Annotations produce:
    - Creation of thread teams
    - Execution in parallel
- Primitives for synchronization
  - Locks
  - Critical sections
  - Single threaded regions
- Primitives for variable sharing
  - Private or Shared
  - Variable initialization for Parallel Regions
Multi-core Architectures

• Dual-core AMD Opteron Processors (type B)
• Sun UltraSPARC IV, Intel Woodcrest Processors (type C)
• Sun UltraSPARC T1 (type D)

Multi-threaded Processors

- **Coarse Grained**
  - Thread Owns the Pipeline
  - Switched out on a stall
  - Easy to implement (low complexity)
  - Poor performance

- **Simultaneous Multithreading (SMT)**
  - Instructions from multiple threads issued in a single clock cycle
  - Hyperthreading

- **Vertical Threading (VT)**
  - Instructions from the same thread issued in a single clock cycle
  - Sun UltraSPARC T1
Page Table Architecture

- Page walk may take up to two memory accesses
- May be a substantial overhead
- TLB misses may be expensive
Intra-node Communication

• Reductions, barriers, etc.
  – Need communication buffers
  – Buffers are small (typically < 1KB)

• SCASH uses Myrinet (through Score) for communication
  – Only need to use intra-node
  – Can be implemented through a shared buffer
  – Use 4KB pages (small communication)
  – Requires a copy
  – Communication queue depth of 32 1KB messages
Memory Protection

- SCASH DSM uses page protection for coherency
- Uses memory faults/handler to trap accesses
- We only use Omni/SCASH on an intra-node system
- Underlying hardware responsible for coherency
- Setting memory protections and servicing the handler adds considerable overhead
- Not needed for an OpenMP application on a shared memory system
- Memory Protections disabled in our design
## Processor TLB Sizes and Coverage

<table>
<thead>
<tr>
<th>ITLB (4KB) Size</th>
<th>Xeon</th>
<th>Opteron</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1DTLB (4KB) Size</td>
<td>128</td>
<td>32</td>
</tr>
<tr>
<td>L1DTLB (2MB) Size</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>L2DTLB (4KB) Size</td>
<td>-</td>
<td>512</td>
</tr>
<tr>
<td>L2DTLB (2MB)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>L2DTLB (4KB) Coverage</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>L2DTLB (2MB) Coverage</td>
<td>64MB</td>
<td>16MB</td>
</tr>
</tbody>
</table>
Scalability - FT

FT

Number of threads

Time (seconds)

Opteron-4KB
Opteron-2MB
Xeon-4KB
Xeon-2MB
Fork/Join Model

master thread

farm of threads

fork for parallel section

join

nested parallelism

implicit barrier

0 1 2 n

n₀  n₁  n₂
Improving the Scalability of OpenMP Applications

- Multi-core architectures are being widely deployed
  - Chip level Multi-threading (CMT)
    - Opteron, Intel Xeon, Sun Niagra
  - Simultaneous Multi-threading (SMT)
    - Intel Xeon and Sun Niagra
  - CMT+SMT
    - Intel Xeon and Sun Niagra
### Application Memory Footprint

<table>
<thead>
<tr>
<th></th>
<th>Instruction</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT (B)</td>
<td>1.6MB</td>
<td>371MB</td>
</tr>
<tr>
<td>CG (B)</td>
<td>1.4MB</td>
<td>725MB</td>
</tr>
<tr>
<td>FT (B)</td>
<td>1.4MB</td>
<td>2.4GB</td>
</tr>
<tr>
<td>SP (B)</td>
<td>1.6MB</td>
<td>387MB</td>
</tr>
<tr>
<td>MG (B)</td>
<td>1.4MB</td>
<td>884MB</td>
</tr>
</tbody>
</table>

- Instruction binary footprint are all less than 2MB
- Data footprint much larger
- ITLB coverage for instruction data is large with 4KB
- Instruction locality high (most time spent in parallel loops)
FT

Number of threads

Time (seconds)

One Core  Two Cores  Four Cores  Eight Hyperthread