How to scale Nested OpenMP Applications on the ScaleMP vSMP Architecture

Dirk Schmidl, Christian Terboven, Andreas Wolf, Dieter an Mey, Christian Bischof

IEEE Cluster 2010 / Heraklion
September 21, 2010
Agenda

- **ScaleMP Overview**
- **Kernel Benchmarks**
  - Page Access Benchmark
  - STREAM
  - SMXV
  - Memory Allocation Tests
  - EPCC Microbenchmarks
- **Real World Applications**
  - Flexible Image Retrieval Engine (FIRE)
  - Simulator for Heat and Mass Transport (SHEMAT-Suite)
- **Conclusions**
1. ScaleMP
   1. 13 board connected via Infiniband
   2. each 2 x Intel Xeon E5420 @ 2,5 GHz
   3. cache coherency by virtualization software (vSMP)
   4. $13 \times 16 = 208$ GB RAM
      ~38 GB reserved for vSMP = **170 GB** available

2. Tigerton (Fujitsu-Siemens RX600): (reference system)
   1. 4 x Intel Xeon X7350 @ 2,93 GHz
   2. 1 x 64 GB RAM
### Page Access Benchmark

**Benchmarks to measure page access time:**

1. **read a remote page**
   - `read_from_other` 43.37 µs 2.97 µs
2. **write a remote page**
   - `write_from_other` 40.44 µs 2.13 µs
3. **write a local page**
   - `write_self` 2.34 µs 1.64 µs

- remote accesses 15 - 20 X more expensive on ScaleMP
- local accesses not affected by vSMP Software

**results in microseconds**
default:
- memory bandwidth not constant over time
- threads are slowly migrated to other boards
- reached maximum ~45 GB/s

bound:
- reached bandwidth ~95 GB/s
- maximum reached from beginning

apply fix thread placement

![Graph showing STREAM - Triad performance comparison between default and bound modes. The default mode shows memory bandwidth not constant over time, requiring slow migration of threads to other boards, and reaching a maximum of ~45 GB/s. The bound mode, however, reaches a bandwidth of ~95 GB/s and has a maximum reached from the beginning.](image-url)
default:
- memory bandwidth not constant over time
- threads are slowly migrated to other boards
- reached maximum ~45 GB/s

bound:
- reached bandwidth ~95 GB/s
- maximum reached from beginning

Conclusion:
- high bandwidth possible for aligned memory accesses
- thread placement is important on ScaleMP
static schedule:
- high performance compared to SMP system
- load imbalance not addressed
guided schedule:
• performance improvement on the SMP system
• bad performance on ScaleMP
• guided schedule introduces high number of remote accesses
sorted schedule:
- nearly no changes on the SMP
- good performance on ScaleMP +10X
sorted schedule:
• nearly no changes on the SMP
• good performance on ScaleMP

Conclusion:
• high bandwidth also for sparse matrixes possible
• dynamic access patterns are a problem
• precomputed pattern needed
Memory Allocation Test

- time (in sec.) to allocate and initialize a 15 GB array in parallel

Default Pages (4 KB):
- allocation takes much longer on ScaleMP
- more Threads introduce larger overhead

Huge Pages (2 MB):
- nearly no benefit for one thread on ScaleMP
- scales better, but still not very well

<table>
<thead>
<tr>
<th></th>
<th>default pages</th>
<th>huge pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tigerton</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Thread</td>
<td>34,79</td>
<td>27,01</td>
</tr>
<tr>
<td>8 Threads</td>
<td>16,6</td>
<td>20,34</td>
</tr>
<tr>
<td>16 Threads</td>
<td>26,9</td>
<td>15,87</td>
</tr>
<tr>
<td><strong>ScaleMP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Thread</td>
<td>211,13</td>
<td>196,29</td>
</tr>
<tr>
<td>8 Threads</td>
<td>423,4</td>
<td>187,45</td>
</tr>
<tr>
<td>16 Threads</td>
<td>449,01</td>
<td>179,25</td>
</tr>
<tr>
<td>104 Threads</td>
<td>432,47</td>
<td>121,76</td>
</tr>
</tbody>
</table>
Memory Allocation Test

- time (in sec.) to allocate and initialize a 15 GB array in parallel

Default Pages (4 KB):
- allocation takes much longer on ScaleMP
- more Threads introduce larger overhead

Huge Pages (2 MB):
- nearly no benefit for one thread on ScaleMP
- scales better, but still not very well

<table>
<thead>
<tr>
<th></th>
<th>default pages</th>
<th>huge pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tigerton</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Thread</td>
<td>34,79</td>
<td>27,01</td>
</tr>
<tr>
<td>8 Threads</td>
<td>16,6</td>
<td>20,34</td>
</tr>
<tr>
<td>16 Threads</td>
<td>26,9</td>
<td>15,87</td>
</tr>
<tr>
<td><strong>ScaleMP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Thread</td>
<td>211,13</td>
<td>196,29</td>
</tr>
<tr>
<td>8 Threads</td>
<td>423,4</td>
<td>187,45</td>
</tr>
<tr>
<td>16 Threads</td>
<td>449,01</td>
<td>179,25</td>
</tr>
<tr>
<td>104 Threads</td>
<td>432,47</td>
<td>121,76</td>
</tr>
</tbody>
</table>

Conclusion:
- memory allocation takes longer on ScaleMP
- codes with a lot of allocation and deallocation of memory will most likely run bad on these systems
### EPCC Microbenchmarks

<table>
<thead>
<tr>
<th>ScaleMP</th>
<th>parallel</th>
<th>for</th>
<th>reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Threads on 1 boards</td>
<td>1.89</td>
<td>0.62</td>
<td>2.07</td>
</tr>
<tr>
<td>2 Threads on 2 boards</td>
<td>297.06</td>
<td>279.91</td>
<td>362.76</td>
</tr>
<tr>
<td>16 Threads on 2 boards</td>
<td>768.28</td>
<td>571.10</td>
<td>923.30</td>
</tr>
<tr>
<td>104 Threads on 13 boards</td>
<td>1453.30</td>
<td>1198.02</td>
<td>3185.69</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tigerton</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Threads</td>
<td>2.14</td>
<td>0.89</td>
<td>2.32</td>
</tr>
<tr>
<td>8 Threads</td>
<td>4.10</td>
<td>1.58</td>
<td>4.40</td>
</tr>
<tr>
<td>16 Threads</td>
<td>6.76</td>
<td>2.25</td>
<td>7.41</td>
</tr>
</tbody>
</table>

- ~100 – 200 X more synchronization time when multiple boards are used

overhead in microseconds
### EPCC Microbenchmarks

<table>
<thead>
<tr>
<th>ScaleMP</th>
<th>parallel</th>
<th>for</th>
<th>reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Threads on 1 boards</td>
<td>1,89</td>
<td>0,62</td>
<td>2,07</td>
</tr>
<tr>
<td>2 Threads on 2 boards</td>
<td>297,06</td>
<td>279,91</td>
<td>362,76</td>
</tr>
<tr>
<td>16 Threads on 2 boards</td>
<td>768,28</td>
<td>571,10</td>
<td>923,30</td>
</tr>
<tr>
<td>104 Threads on 13 boards</td>
<td>1453,30</td>
<td>1198,02</td>
<td>3185,69</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tigerton</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Threads</td>
<td>2,14</td>
<td>0,89</td>
<td>2,32</td>
</tr>
<tr>
<td>8 Threads</td>
<td>4,10</td>
<td>1,58</td>
<td>4,40</td>
</tr>
<tr>
<td>16 Threads</td>
<td>6,76</td>
<td>2,25</td>
<td>7,41</td>
</tr>
</tbody>
</table>

- ~100 – 200 X more synchronization time when multiple boards are used.

**Conclusion:**
- Use as few synchronizations as possible.
- If synchronizations are your performance bottleneck, then your code will most likely not run well on ScaleMP.
Flexible Image Retrieval Engine (FIRE)

FIRE = Flexible Image Retrieval Engine
- Compare the performance of common features on different databases
- Analysis of correlation of different features

Thomas Deselaers and Daniel Keysers, RWTH I6: Chair for Human Language Technology and Pattern Recognition
Flexible Image Retrieval Engine (FIRE)

\[
D(Q, X) := \sum_{m=1}^{M} w_m \cdot d_m(Q_m, X_m)
\]

- Q: query image, X: set of database images
- Qm, Xm: m-th feature of Q and X
- \(d_m\): distance measure, \(w_m\): weighting coefficient
- Return the k images with lowest distance to query image

- Well-suited for Shared-Memory parallelization:
  Data Mining in a large image database!

- Two levels of parallelism:
  - Process multiple query images in parallel
  - Process database comparison for one query image in parallel
Flexible Image Retrieval Engine (FIRE)

- $|Q| = 18$
- $|X| = 1000$

- Speedup of ~80 on 104 Cores
- Small load imbalance due to small number of query images
Simulator for Heat and Mass Transport (SHEMAT-Suite)

Geothermal Simulation Package to simulate groundwater flow, heat transport, and the transport of reactive solutes in porous media at high temperatures (3D)

Applied Geophysics and Geothermal Energy, E.ON Energy Research Center

Written in Fortran, two levels of parallelism:

- Independent Computations of the Directional Derivatives using AD
- Setup and Solving of linear equation systems
Simulator for Heat and Mass Transport (SHEMAT-Suite)

- Binding nested OpenMP programs was a problem, since there is no way to specify the mapping of inner teams to cores

=> implement manual thread binding

<table>
<thead>
<tr>
<th>Number of Boards</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>ser 1 2 3 4 5 6 7 8 9 10 11 12 13</td>
<td></td>
</tr>
</tbody>
</table>

- original, unbound (1x8)
- original, unbound (8x1)
- original, bound (1x8)
- original, bound (8x1)

**ScaleMP**

- Boards
- Cores
Simulator for Heat and Mass Transport (SHEMAT-Suite)

- creation of private arrays turned out to be the performance bottleneck

=> changed data management to use slices of a larger global array
Estimated speedup
number of directional derivatives, 104 Cores used:

First Phase: 
8 Threads working per board
Best result for 8x1 on one board is 5242.67s \(\equiv\) 131.07 s/8 derivatives

Second Phase: 
1 Threads working per board
Best result for 1x1 on one board is 19404.3s \(\equiv\) 60.64 s/derivative
Simulator for Heat and Mass Transport (SHEMAT-Suite)

Estimated speedup:

\[
131.07s + 131.07s + 131.07s + 60.64s = 453.85s
\]

First Phase: 8 Threads working per board
Best result for 8x1 on one board is 5242.67s \(\cong\) 131.07 s/derivative

Second Phase: 1 Threads working per board
Best result for 1x1 on one board is 19404.3s \(\cong\) 60.64 s/derivative

Speedup \(\cong\) 42.76
Simulator for Heat and Mass Transport (SHEMAT-Suite)

Best Speedup: 41.48
Estimated theoretical Speedup: 42.76
Simulator for Heat and Mass Transport (SHEMAT-Suite)

ScaleMP-N:

1. 16 boards each 2 x Intel Xeon E5550 @ 2,67 GHz (Nehalem)
2. 320 GB RAM - ~32 GB reserved for vSMP = 288 GB available
Summary

- high remote memory access (20X) and synchronization (150X) time
- high memory bandwidth for dense (~95 GB/s) and sparse matrices (10X)
  ⇒ distinct cc-NUMA behavior

- FIRE achieved speedup of ~80
- SHEMAT-Suite achieved speedup of 41.5 (theoretical maximum 42.7)
  ⇒ Real user applications can profit from the ScaleMP architecture.
Conclusion

Provided that your application …

... takes care about thread placement...  
... takes care about data placement...  
... avoids too many synchronizations...  
... reduces the number of memory allocations ...

... than the ScaleMP architecture can deliver a very good performance and a large amount of shared memory at a low price point.
Thank you for your attention!

Questions?