OpenMP on cc-NUMA Architectures

Christian Terboven, Dieter an Mey, Dirk Schmidl
{terboven, anmey, schmidl}@rz.rwth-aachen.de

Center for Computing and Communication
RWTH Aachen University

7th Intel EMEA HPC Roundtable
April 14th, Paris, France
Agenda

- Introduction and Motivation
- Explicit Thread and Data Placement
- Issues with Multi-level Parallelism
- Summary and Outlook
The OpenMP Memory Model

- One Shared Memory, Private Memory per thread → equal distance from each core to the whole memory assumed
cc-NUMA
- It is hard (= expensive) to build large SMPs offering a flat memory architecture with low latency and high bandwidth
- Large Shared-Memory systems have physically distributed memories connected by a cache-coherent interconnect
- It is profitable (= important) to keep threads close to data

CMP: chip multi-processing
- Recent processors contain multiple cores sharing the path to the memory, different cache layouts possible

CMT/SMT: chip multi-threading, simultaneous multi-thr.
- Some cores are capable of executing multiple instructions streams (= HW threads) (quasi) simultaneously
Three levels of caches:
- L1 per core: 32 KB
- L2 per core: 256 KB
- L3 per socket: 8192 KB

Four cores per socket: Each is capable of executing two threads simultaneously.
2-socket system from Sun

Node #0 (17 GB)

Socket #0
- L3 (8192 KB)
- L2 (256 KB)
- L1 (32 KB)
- Core #0: 0, 8
- Core #1: 1, 9
- Core #2: 2, 10
- Core #3: 3, 11

Node #1 (17 GB)

Socket #1
- L3 (8192 KB)
- L2 (256 KB)
- L1 (32 KB)
- Core #0: 4, 12
- Core #1: 5, 13
- Core #2: 6, 14
- Core #3: 7, 15

2-socket system from HP

Node #1 (23 GB)

Socket #1
- L3 (8192 KB)
- L2 (256 KB)
- L1 (32 KB)
- Core #0: 0, 8
- Core #2: 2, 10
- Core #1: 4, 12
- Core #3: 6, 14

Node #0 (23 GB)

Socket #0
- L3 (8192 KB)
- L2 (256 KB)
- L1 (32 KB)
- Core #0: 1, 9
- Core #1: 3, 11
- Core #2: 5, 13
- Core #3: 7, 15
• Systems exhibit slight differences that may have a huge impact on the performance
• Do not hard-code architecture details in your app, instead follow a strategy-based approach where possible
• Intel Compilers: KMP_AFFINITY environment variable controls how threads are placed, i.e. compact versus scatter
OpenMP on cc-NUMA Architectures

14.04.2010 – C. Terboven

Agenda

- Introduction and Motivation
- Explicit Thread and Data Placement
- Issues with Multi-level Parallelism
- Summary and Outlook
Current operating systems employ the first-touch strategy: Data is placed closest to the thread that touches it first.

The OS may do a reasonable job,
- if the machine is not overloaded,
- and the first-touch policy has been carefully taken into account,
- and the program does not change the memory access pattern.

There are options for additional user control:
- Explicit binding of threads to processors:
  - By environment variables, i.e. \texttt{KMP\_AFFINITY}
  - By user commands
    - Linux: \texttt{taskset}, \texttt{numactl}
    - Windows: \texttt{start /affinity}
User Control of Affinity (2/2)

There are options for additional user control:

- Explicit binding of threads to processors (cont’d):
  - By system calls
    - Linux: `sched_setaffinity()`
    - Windows: `SetThreadAffinityMask()`
  - Explicit numa-aware memory allocation:
    - By carefully touching data by the thread which later uses it
    - By changing the default memory allocation strategy
      - Linux: `numactl` command
      - Windows: `VirtualAllocExNuma()` (limited functionality)
    - By explicit migration of memory pages
      - Linux: `move_pages()`
      - Windows: no option
STREAM-Triad on Nehalem-EX (1/2)

- Serial init
- Serial init + numactl --interleave
- Parallel init

<table>
<thead>
<tr>
<th># Threads</th>
<th>Memory Bandwidth [MB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000</td>
</tr>
<tr>
<td>2</td>
<td>20000</td>
</tr>
<tr>
<td>4</td>
<td>30000</td>
</tr>
<tr>
<td>8</td>
<td>40000</td>
</tr>
<tr>
<td>16</td>
<td>50000</td>
</tr>
<tr>
<td>32</td>
<td>60000</td>
</tr>
<tr>
<td>48</td>
<td>70000</td>
</tr>
<tr>
<td>64</td>
<td>80000</td>
</tr>
</tbody>
</table>
STREAM-Triad on Nehalem-EX (2/2)

- Missing the parallel initialization can spoil all the fun ...

```c
// allocation of arrays
double *a, *b, *c;
a, b, c = (double*) malloc(N*sizeof(double));
// parallel initialization: data allocated where used
#pragma omp parallel for
for (i=0; i<N; i++) a[i]=...=0.0;
// triad-ing with optimal memory placement
#pragma omp parallel for
for (i=0; i<N; i++) a[i]=b[i]+scalar*c[i];
```

- If data is setup in serial, but the computation is parallel, the data to thread affinity may be very bad
  - Either take care explicitly by binding + first-touch parallel init
  - Or apply random / round robin data placement

- Support by current multi-threading paradigms is missing!
A first Summary

- Everything under control?
- In principle Yes, but only if
  - threads can be bound explicitly,
  - data can be placed well by first-touch, or can be migrated,
  - you focus on a specific platform (= os + arch) $\rightarrow$ no portability

- What if the data access pattern changes over time?

- What if you use more than one level of parallelism?
SPH

- SPH = Smoothed Particle Hydrodynamics, fluid simulation
- Water is represented as a set of (many) particles
- Domain / Grid changes over time ...

Every color represents one thread being executed on one socket

Green = local access
Red = remote access

cc-NUMA ignored
One-time data distrib.
Many-time data distrib.
Agenda

- Introduction and Motivation
- Explicit Thread and Data Placement
- Issues with Multi-level Parallelism
- Summary and Outlook
The OpenMP runtime organizes the (OS) threads in a pool.
OpenMP Nested (2/3)

- Experimental extension implemented in Sun Studio compilers

- SUNW_MP_PROCBIND=true
  Binds threads to cores in the order in which they were created (similar features are available for almost all OpenMP compilers)

- SUNW_MP_THR_AFFINITY=true
  Maintains thread assignments (on the cost of possible resource waste)
In combination with both environment variables a short loop takes care of associating the threads of the inner teams to cores in the same locality domain.

```
do i = 1, 4
  !$omp parallel num_threads(i)
  !$omp parallel num_threads(2)
  continue
  !$omp end parallel
  !$omp end parallel
```
Nested STREAM-Triad

<table>
<thead>
<tr>
<th>#thrds</th>
<th>First touch</th>
<th>Affinity</th>
<th>min MB/s</th>
<th>max MB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 8</td>
<td>Initial thread</td>
<td>n.a.</td>
<td>2525</td>
<td>2534</td>
</tr>
<tr>
<td>1 x 8</td>
<td>All inner threads</td>
<td>n.a.</td>
<td>11786</td>
<td>11869</td>
</tr>
<tr>
<td>4 x 2</td>
<td>Initial thread</td>
<td>no</td>
<td>4x629</td>
<td>4x631</td>
</tr>
<tr>
<td>4 x 2</td>
<td>Initial thread</td>
<td>yes</td>
<td>4x628</td>
<td>4x631</td>
</tr>
<tr>
<td>4 x 2</td>
<td>Inner master</td>
<td>no</td>
<td>4x1312</td>
<td>4x1332</td>
</tr>
<tr>
<td>4 x 2</td>
<td>Inner master</td>
<td>yes</td>
<td>4x1329</td>
<td>4x1334</td>
</tr>
<tr>
<td>4 x 2</td>
<td>Inner master</td>
<td>Yes+sort</td>
<td>4x2881</td>
<td>4x2943</td>
</tr>
<tr>
<td>4 x 2</td>
<td>All inner threads</td>
<td>no</td>
<td>4x2640</td>
<td>4x2948</td>
</tr>
<tr>
<td>4 x 2</td>
<td>All inner threads</td>
<td>yes</td>
<td>4x2893</td>
<td>4x2950</td>
</tr>
<tr>
<td>4 x 2</td>
<td>All inner threads</td>
<td>Yes + sort</td>
<td>4x2922</td>
<td>4x2934</td>
</tr>
</tbody>
</table>

SunFire V40z dualcore
Binding = 0,1,2...7
empty machine
These issues become more severe on larger machines...

The ScaleMP vSMP software employs virtualization techniques to run a single system image (Linux) on a set of nodes connected via InfiniBand.

The machine at RWTH Aachen University:
- 13 boards connected via InfiniBand
- 13x 2x Intel Xeon E5420 (Harpertown, 2.5 GHz)
- 13x 16 GB of main memory
  - 38 GB used for management and automatic page migration
  - 170 GB available for applications
Application Case Study: SHEMAT-Suite

- Geothermal Simulation Package to simulate groundwater flow, heat transport, and the transport of reactive solutes in porous media at high temperatures (3D)
  - Forward simulation
    - 3D finite-differences solver, Coupled transient equations for groundwater flow, Compute state variables from rock properties
  - Inverse computation
    - Parameter estimation

- Written in Fortran, two levels of parallelism
  - Independent Computations of the Directional Derivatives
  - Setup and Solving of linear equation systems
Optimizations:
1. Binding threads to let inner teams run on one board.
2. Avoiding unnecessary temporal data usage.
Agenda

- Introduction and Motivation
- Explicit Thread and Data Placement
- Issues with Multi-level Parallelism
- Summary and Outlook
The final Summary

- Affinity support is essential for OpenMP (or any other Shared-Memory paradigm) on cc-NUMA machines!

- Using OpenMP 3.0 on Linux it is possible to solve most issues

- Windows offers less functionality

- Open issues:
  - Better support for Nested Parallelism
  - Interaction with Resource Management Systems
    - Make the application aware of its intended part of the machine
    - Provide guidance for the (OpenMP) runtime
Our favorite proposal:

1. Do not describe the HW architecture within OpenMP
2. Provide means to describe the structure of a (nested) OpenMP program as a series of thread trees
3. Provide means to pass this information along with some guidance in form of strategies to the runtime
4. Allow the runtime to take care of the mapping of threads to cores / the HW (analogous to the MPI topology concept)
5. Guarantee threadprivate persistence as long as the tree remains constant
6. Provide means for first-touch / random placement and migration to enable portability
The End

Thank you for your attention!
Backup: Compact vs. Scatter on Nehalem

- spec_rv1_omp_ammp
  - nehalem x5570 (2.93 GHz/intel 11.0.69, scatter binding)
  - nehalemex x7650 (2.27 GHz, intel 11.1, scatter binding)
  - nehalemex x7650 (2.27 GHz, intel 11.1, compact binding)

- spec_rv1_omp_apsi
  - nehalem x5570 (2.93 GHz/intel 11.0.69, scatter binding)
  - nehalemex x7650 (2.27 GHz, intel 11.1, scatter binding)
  - nehalemex x7650 (2.27 GHz, intel 11.1, compact binding)

- spec_rv1_omp_applu
  - nehalem x5570 (2.93 GHz/intel 11.0.69, scatter binding)
  - nehalemex x7650 (2.27 GHz, intel 11.1, scatter binding)
  - nehalemex x7650 (2.27 GHz, intel 11.1, compact binding)

- spec_rv1_omp_art
  - nehalem x5570 (2.93 GHz/intel 11.0.69, scatter binding)
  - nehalemex x7650 (2.27 GHz, intel 11.1, scatter binding)
  - nehalemex x7650 (2.27 GHz, intel 11.1, compact binding)