Scalable Shared Memory Programming with OpenMP

and Current Trends …

Workshop on Large-Scale Computer Simulation
March 9-11, 2001
Aachen / Jülich

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

- OpenMP in a Nutshell
- Scalable OpenMP Programming
- Hybrid Parallelization
- New Features in OpenMP 3.0 / 3.1
- Towards OpenMP 4.0
- Summary
Overview

- OpenMP in a Nutshell
- Scalable OpenMP Programming
- Hybrid Parallelization
- New Features in OpenMP 3.0 / 3.1
- Towards OpenMP 4.0
- Summary
OpenMP – What is it about?

- OpenMP is an Application Program Interface (API) for
  - explicit
  - portable
  - shared-memory parallel programming
  - in C/C++ and Fortran.

- OpenMP consists of
  - compiler directives,
  - runtime calls and
  - environment variables.

- Today it is supported by all major compilers on Unix and Windows platforms
  - GNU, IBM, Oracle, Intel, PGI, Absoft, Lahey/Fujitsu, PathScale, HP, MS, Cray

http://openmp.org/wp/openmp-specifications/
OpenMP - Organisations

- **OpenMP Architecture Review Board**
  - Non-profit corporation which owns the OpenMP brand and controls the specification
  - Directors: Josh Simons (VMware), Sanjiv Shah (Intel), Koh Hotta (Fujitsu)
  - CEO: Larry Meadows (Intel)

- **OpenMP Language Committee**
  - Works on the specification

- **OpenMP User Community – cOMPunity**
  - cOMPunity has one vote in the ARB
  - Non-ARB-members are invited to contribute through cOMPunity

- **Int’l Workshop on OpenMP (IWOMP)**
  - Annual OpenMP Workshop organized by cOMPunity and the ARB
  - IWOMP 2011, June 13-15 in Chicago, USA

[www.openmp.org](http://www.openmp.org)
[www.iwomp.org](http://www.iwomp.org)
[www.compunity.org](http://www.compunity.org)
<table>
<thead>
<tr>
<th>Year</th>
<th>IBM</th>
<th>KAI/Intel</th>
<th>SGI</th>
<th>Fujitsu</th>
<th>NEC</th>
<th>SUN/Oracle</th>
<th>IBM</th>
<th>KAI/Intel</th>
<th>SGI</th>
<th>Fujitsu</th>
<th>NEC</th>
<th>SUN/Oracle</th>
<th>IBM</th>
<th>KAI/Intel</th>
<th>SGI</th>
<th>Fujitsu</th>
<th>NEC</th>
<th>SUN/Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
<tr>
<td>2001</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
<tr>
<td>2002</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
<tr>
<td>2003</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
<tr>
<td>2004</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
<tr>
<td>2005</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
<tr>
<td>2006</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
<tr>
<td>2007</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
<tr>
<td>2008</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
<tr>
<td>2009</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
<tr>
<td>2010</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
<td>IBM</td>
<td>KAI/Intel</td>
<td>SGI</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>SUN/Oracle</td>
</tr>
</tbody>
</table>
OpenMP - History

- October 1997 OpenMP version 1.0 for Fortran.
- October 1998 OpenMP version 1.0 for C/C++.
  - November 2000 OpenMP version 2.0 for Fortran.
  - March 2002 OpenMP version 2.0 for C/C++.
  - May 2005 OpenMP version 2.5 combined for C/C++ and Fortran
- May 2008 OpenMP Version 3.0 for C/C++ and Fortran
- February 2011 OpenMP Draft Version 3.1 for public comment
Fork-join model of parallel execution

Parallel regions are executed (redundantly) by a team of threads.

Work can be distributed among the threads of a team by worksharing constructs like the parallel loop construct, which provides powerful scheduling mechanisms.

Since V3.0 (2008) Tasks (code plus data) can be enqueued by a task construct and their execution by any thread of the team can be deferred.

Support for Nested parallelism has been improved with V3.0.
OpenMP in a Nutshell
Memory Model

- **Shared-Memory model**
  - All threads share a common address space (shared memory)
  - Threads can have private data

- **Relaxed memory consistency**
  - Temporary View ("Caching"): Memory consistency is guaranteed only after synchronization points, namely implicit and explicit **flushes**
    - Each OpenMP **barrier** includes a **flush**
    - Exit from worksharing constructs include barriers by default (**but not entries!**)
    - Entry to and exit from **critical regions** include a **flush**
    - Entry to and exit from lock routines (OpenMP API) include a **flush**
calculate Pi by numerical integration

double f(double x) {
    return (double)4.0 / ((double)1.0 + (x*x));
}

void computePi() {
    double h = (double)1.0 / (double)n;
    double sum = 0, x;

    #pragma omp parallel for schedule(static) \ 
        private(x) shared(h,n) reduction(+:sum)
    for (int i = 1; i <= n; i++) {
        x = h * ((double)i - (double)0.5);
        sum += f(x);
    }

    myPi = h * sum;
}
Overview

- OpenMP in a Nutshell
- Scalable OpenMP Programming
- Hybrid Parallelization
- New Features in OpenMP 3.0 / 3.1
- Towards OpenMP 4.0
- Summary
Increasing Scalability
Extend Parallel Region, Avoid Barriers

\begin{verbatim}
!$omp parallel private(n,m,l,i,j,k,lijk)
do n = 1,7
  do m = 1,7
    !$omp do
    do l = LSS(itsub),LEE(itsub)
      i = IG(l)
      j = JG(l)
      k = KG(l)
      lijk = L2IJK(l)
      RHS(l,m) = RHS(l,m) - &
      FJAC(lijk,lm00,m,n) * DQCO(i-1,j,k,n,NB) * FM00(l) - &
      FJAC(lijk,lp00,m,n) * DQCO(i+1,j,k,n,NB) * FP00(l) - &
      FJAC(lijk,10m0,m,n) * DQCO(i,j-1,k,n,NB) * F0M0(l) - &
      FJAC(lijk,10p0,m,n) * DQCO(i,j+1,k,n,NB) * F0P0(l) - &
      FJAC(lijk,100m,m,n) * DQCO(i,j,k-1,n,NB) * F00M(l) - &
      FJAC(lijk,100p,m,n) * DQCO(i,j,k+1,n,NB) * F00P(l)
    end do
    !$omp do nowait
  end do
end do
!$omp end parallel
\end{verbatim}

Partitioning the long loop

No barrier, zero overhead

Check for correctness!

(Intel Inspector, aka Thread Checker)

\textit{D. an Mey, S. Schmidt: From a Vector Computer to an SMP-Cluster - Hybrid Parallelization of the CFD Code PANTA, EWOMP 2000, Edinburgh}
Increasing Scalability
Orphaning: 1 PR includes 69 Parallel Loops

- Simulation of the heat flow in a rocket combustion chamber
- Finite Element Method
- OpenMP Parallelization
  - 30000 lines of Fortran
  - 200 OpenMP directives, 69 parallel loops,
  - 1 main parallel region
- ~40x Speed-up on 68 UltraSPARC III processors (Sun Fire 15K)

- OpenMP 3.1 Glossary: orphaned construct
  - A construct that gives rise to a region whose binding thread set is the current team, but that is not nested within another construct giving rise to the binding region.

_D. an Mey, T. Haarmann: Pushing Loop-Level Parallelization to the Limit, EWOMP 2002, Rome_
Increasing Scalability
Load Imbalances, Nested Parallelism

- Analysis of complex and accurate fluid dynamics simulations
- Extraction of Critical Points for Virtual Reality (Location with velocity = 0)
- 25-100% efficiency with 128 threads on 72 UltraSPARC IV dual core processors (Sun Fire E25K) depending on data set

```c
// Loop over time levels
#pragma omp parallel for num_threads(nTimeThreads) schedule(dynamic,1)
for (curT=1; curT<=maxT; ++curT) {
    // Loop over Blocks
    #pragma omp parallel for num_threads(nBlockThreads) schedule(dynamic,1)
    for (curB=1; curB<=maxB; ++curB) {
        // Loop over Cells
        #pragma omp parallel for num_threads(nCellThreads) schedule(guided)
        for (curC=1; curC<=maxC; ++curC) {
            FindCriticalPoints (curT, curB, curC); // highly adaptive algorithm (bisectioning)
        }
    }
}

A. Gerndt, S. Sarholz, et.al.: 3-D Critical Points Computed by Nested OpenMP, SC 2006, Tampa
```
Non Uniform Memory Architectures (NUMA)

Sun Fire V40z
one of the early popular NUMA systems
with 4 dual core x86-64 processors

AMD Opteron 875, dual core, 2.2 GHz

Cache-coherent
HyperTransport Connections
Memory Allocation Policy

- If data is setup in serial region, but the computation in parallel regions, the data to thread affinity may hurt performance very badly!
  - Either take care of thread binding explicitly + first-touch parallel initialization
    - or apply random / round robin data placement

    // allocation of arrays
    double *a, *b, *c;
    a, b, c = (double*) malloc(N*sizeof(double));

    // parallel initialization of data where used later on
    #pragma omp parallel for schedule(static)
    for (i=0; i<N; i++) a[i]=…=0.0;

    // calculation with optimal memory placement and identical schedule
    #pragma omp parallel for schedule(static)
    for(i=0; i<N; i++) a[i]=b[i]+scalar*c[i];
Sparse Matrix-Vector Multiplication on NUMA

4 x dualcore Opteron 2.2 GHz, ccNUMA

12x dualcore UltraSPARC 1.2 GHz, flat memory

19,6 Mio nonzeros
233,334 matrix dimension
225 MB memory footprint

C. Terboven, et.al.: Parallelization of the C++ Navier-Stokes Solver DROPS with OpenMP, ParCo 2005, Malaga
Memory Bandwidth on a 4-way Nehalem EX System (Stream Triad)

Here, each Nehalem EX processor has 8 cores and 16 threads which adds up to 32 cores and 64 threads (Intel HyperThreading)
Virtual Shared Memory Processing on an Infiniband-Cluster with ScaleMP

- **SHEMAT-Suite**
  - Geothermal Simulation of CO₂ Storage
  - Simulating Groundwater flow, heat transfer and transport of reactive solutes
  - ~10x speed-up with 2nd level of OpenMP

![Graph showing speedup with number of nodes](image)

- 16 node Nehalem EP Cluster with IB-QDR
- 13 node Harpertown Cluster with IB-DDR
Overview

- OpenMP in a Nutshell
- Scalable OpenMP Programming
- Hybrid Parallelization
- New Features in OpenMP 3.0 / 3.1
- Towards OpenMP 4.0
- Summary
Adding OpenMP to MPI may be beneficial

- XNS (M. Behr, CATS, RWTH)
  - Simulation of Hydro-Dynamic forces of the Ohio Dam
- OpenMP Parallelization:
  - 9 parallel regions
  - Human effort: ~ 6 weeks
- Best MPI performance:
  - 48 nodes, one MPI process per node
- Best Hybrid performance:
  - 32 nodes, one MPI process per node, 4 threads per process
  - 1.5x improvement to MPI-only

PPN = processes per node
TPP = threads per process

Harpertown Cluster with IB-DDR

Best effort hybrid
Best effort MPI only

1 PPN
1 PPN / 4 TPP
2 PPN
2 PPN / 2 TPP
4 PPN
8 PPN
Adding OpenMP to MPI may be beneficial

- XNS (M. Behr, CATS, RWTH)
  - Simulation of Hydro-Dynamic forces of the Ohio Dam
- OpenMP Parallelization:
  - 9 parallel regions
  - Human effort: ~ 6 weeks
- Best absolute MPI performance:
  - 48 nodes, 1 MPI process per node
    - 35.9 sec
- Best absolute Hybrid performance:
  - 16 nodes, one MPI process per socket, 4 threads per process
    - 33.7 sec

![Graph showing performance](image)

Nodes

PPN = processes per node
TPP = threads per process

Nehalem EP Cluster with IB-QDR

PPN=2 TPP=4

RZ: Dieter an Mey  Scalable Shared Memory Programming with OpenMP  Folie 22
Overview

- OpenMP in a Nutshell
- Scalable OpenMP Programming
- Hybrid Parallelization
- New Features in OpenMP 3.0 / 3.1
- Towards OpenMP 4.0
- Summary
New in OpenMP 3.0
Tasks

- Tasks allow to parallelize irregular problems, e.g.
  - unbounded loops
  - recursive algorithms
  - Producer / Consumer patterns
  - and more ...

- Task: A unit of work which can be executed later
  - Can also be executed immediately

- Tasks are composed of
  - Code to execute
  - Data environment
  - Internal control variables (ICV)
Parallelization of an unbounded while loop

- All loop iterations are independent from each other!
- Number of iterations unknown up front
- would have been unconvenient beforehand (inspector/executor method)

```c
typedef list<double> dList;  dList myList;
#pragma omp parallel
{
    #pragma omp single
    {
        dList::iterator it = myList.begin();
        while (it != myList.end())
        {
            #pragma omp task firstprivate(it)
            { *it = processListItem(*it);  }
            it++;
        }
    } // end single
} // end parallel region
```

New in OpenMP 3.0
Tasking Example
New in OpenMP 3.0
Improved Support for Nested Parallelism

- **New runtime functions:**

  ```c
  int omp_get_level()
  // Which current nested level?
  int omp_get_active_level()
  // How many nested active parallel regions (>1 thread)?
  int omp_get_ancestor_thread_num(int level)
  // thread-id of ancestor thread at a given level?
  int omp_get_team_size(int level)
  // Size of ancestor’s team at a given level?
  ```

- **New environment variables (plus corresponding runtime functions):**

  - `OMP_MAX_NESTED_LEVEL` # maximum number of active parallel regions
  - `OMP_THREAD_LIMIT` # maximum total number of OpenMP threads
**Scalable Shared Memory Programming with OpenMP**

- **Static schedule**
  
  ```
  #pragma omp for schedule(static) nowait
  for (i = 1; i < N; i++) a[i] = ...
  #pragma omp for schedule(static)
  for (i = 1; i < N; i++) c[i] = a[i]
  ```

- **Loop collapsing**
  
  ```
  #pragma omp for collapse(2)
  for (i = 1; i < N; i++)
    for (j = 1; j < M; j++)
      foo(i, j);
  ```

- **New variable types allowed in for-Worksharing**
  
  ```
  #pragma omp for
  for (unsigned int i = 0; i < N; i++) foo(i);

  vector v; vector::iterator it;
  #pragma omp for
  for (it = v.begin(); it < v.end(); it++)
    foo(it);
  ```

---

**News in OpenMP 3.0**

- **Miscellaneous**

  - Allowed in OpenMP 3.0 if and only if:
    - Number of iterations is the same
    - Chunksize is the same (or not specified)

  - Iteration space from i-loop and j-loop is collapsed into a single one, if loops are perfectly nested and form a rectangular iteration space.

  - Legal in OpenMP 3.0:
    - Unsigned integer types
    - Pointer types
    - Random access iterators (C++)
Many small corrections and clarifications throughout the whole spec

A tiny step towards improved NUMA support:

- `export OMP_PROC_BIND=true`
  
  # please, don't move OpenMP threads between processes

- `export OMP_NUM_THREADS=4,3,2`
  
  # control thread number for nested parallelism up front

Refinements to the OpenMP Tasking Model:

- The `taskyield` directive denotes a user-defined task scheduling point at which the current task may be suspended (and resumed later).

- The `mergeable` clause indicates that the task may have the same data region as the generating task region.

- The `final` clause denotes all descendent tasks to be executed sequentially in the same region (immediate execution).
More miscellaneous extensions:

- The atomic construct now accepts the clauses read, write, update and capture to ensure atomicity of the corresponding operations.

- The firstprivate clause accepts const-qualified types in C/C++ and intent(in) declared types in Fortran.

- For C/C++ the reduction clause now also accepts min and max reductions for built-in datatypes, still excluding aggregate types, pointer types, and reference types.

- The new omp_in_final() API routine allows to determine whether the calling task is final.
Overview

- OpenMP in a Nutshell
- Scalable OpenMP Programming
- Hybrid Parallelization
- New Features in OpenMP 3.0 / 3.1
- Towards OpenMP 4.0
- Summary
Towards OpenMP 4.0

Overall Goals

- Error Model
- Interoperability and Composability
- NUMA Support ("Affinity")
- Accelerators
- Tasking Extensions
Towards OpenMP 4.0

OpenMP Error Model

- C, C++ and Fortran suggest different approaches: Error Codes, Error Variables, Call Backs, Exceptions, ...

- First step: Being able to react to an error.

- Current plan: Introduction of a directive to end the execution of OpenMP constructs and definition of Cancellation points

  ```c
  #pragma omp done [scope]
  ```

  - To end the current Parallel Region
  - To end the current Worksharing construct
  - To end the current Task

- Pre-defined as well as user-defined Cancellation points at which the execution is guaranteed to end
Towards OpenMP 4.0
NUMA Support

<table>
<thead>
<tr>
<th>Issue / Ticket</th>
<th>Example</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controlling the Number of Threads on Multiple ... Levels</td>
<td>export OMP_NUM_THREADS=4,3,2</td>
<td>3.1</td>
</tr>
<tr>
<td>Controlling Thread Binding</td>
<td>export OMP_PROC_BIND=TRUE</td>
<td>3.1</td>
</tr>
<tr>
<td>Restricting the Processor Set for Program Execution</td>
<td>setenv OMP_PROCSET 0,2,4,6, 8,10, 12,14</td>
<td>4.x</td>
</tr>
<tr>
<td>Controlling the Placement of Threads within the Processor Set</td>
<td>export OMP_AFFINITY=scatter,,compact !$omp parallel affinity( scatter )</td>
<td>4.x</td>
</tr>
<tr>
<td>Controlling the Initial Placement of Shared Data</td>
<td>export OMP_MEMORY_PLACEMENT=spread</td>
<td>4.x</td>
</tr>
<tr>
<td>Adapting the Placement of Shared Data at Runtime</td>
<td>!$omp migrate[(variable list)] strategy( ...)</td>
<td>4.x</td>
</tr>
<tr>
<td>Distance Matrix</td>
<td>?</td>
<td>4.x</td>
</tr>
</tbody>
</table>
Accelerator Subcommittee led by James Beyer (Cray) is very active.

Extensions to the Execution and Memory Model

- *Accelerator Tasks* can be created to execute an *Accelerator Region*
- Data can reside on the *Host*, the *Accelerator Device*, or both.
  - Directives control data transfer
  - Details are left to the runtime

Accelerator Execution Region

- Marks the code to be executed on an accelerator

Accelerator Data Region

- Define the data scope to be reused across multiple accelerator regions
Feedback from the user community:

- Tasks need *Reductions*
- Tasks need *Dependencies*

There is currently no way to identify tasks (and it is not intended to create one), but we need a facility to denote tasks belonging together.

Current approach: *Taskgroup*

- Defined as a structured block, an OpenMP Region
- Reductions may be performed inside a Taskgroup

Current approach regarding dependencies: Expression via addresses, thus Array Shaping Expressions are necessary.
Overview

- OpenMP in a Nutshell
- Scalable OpenMP Programming
- Hybrid Parallelization
- New Features in OpenMP 3.0 / 3.1
- Towards OpenMP 4.0
- Summary
OpenMP scales
- within the node (there is a lot of resource sharing, though)
- if you do it right (extend parallel regions, try to avoid barriers …)
- Consider data-thread-affinity on NUMA, use OS tools for control
- Beware of data races – there are verification tools (like Intel Inspector)

OpenMP may even scale across nodes (ScaleMP)

OpenMP works well together with MPI
- Frequent sweet spot: one MPI process per socket, one thread per core
- Again: Consider data-thread-affinity on NUMA
  (Depends on MPI implementation and resource management system)

OpenMP progresses slowly
- OpenMP is closely tight to into the base languages which makes it tough
- Stay tuned for OpenMP on accelerators
Monday, March 21, afternoon
Announcement of the upcoming RWTH Compute Cluster with renowned Speakers from Bull, Intel, GRS, and Oracle

Tuesday, March 22 – Thursday, March 24,
Tutorials in Serial, OpenMP and MPI Programming

Friday, March 25
GPGPU Programming with Michael Wolfe (PGI)

PPCES, March 21-25, 2010, Aachen

www.rz.rwth-aachen.de/ppces