Multicore-aware parallelization strategies for efficient temporal blocking
(BMBF project: SKALB)

Department für Informatik
HPC Services, Regionales Rechenzentrum Erlangen
Friedrich-Alexander-Universität Erlangen-Nürnberg
SKALB project survey

- Partners:
  - TU Braunschweig
  - TU Dortmund / IANUS
  - HLRS Stuttgart
  - Erlangen: LSS / RRZE (project lead)

- Ass. partners: Intel, CRAY, BASF, Sulzer, hhpberlin, HP

- Goal: Efficient implementation of Lattice-Boltzmann CFD solvers for complex multi-physics applications on petascale supercomputers

- Main tasks of RRZE:
  - Multicore-specific implementation & optimization of kernels & full applications.
  - Static domain decomposition
  - Benchmarking and showcases

  (BMBF Call „HPC-Software für skalierbare Parallelrechner“)

  Funding: 1,8 M€
Multicore-aware parallelization approach:

- Efficient, parallel, multicore-aware temporal blocking
- Proof of concept: Simple iterative method w. regular stencil updates (Jacobi)
- Elaborate spatial blocking techniques have been implemented (not shown)
- Socket-/node level implementation + multi-halo data exchange for multi-node computations

The multicore (r)evolution
A standard compute node

2-socket shared-memory node of cache-coherent
Non-Uniform Memory Access (ccNUMA) type

Shared Caches:
- Data exchange
- Thread synchronization

- Data Coherency!
- L3 cache traffic?
- Scalable?
- MPI parallel?

<table>
<thead>
<tr>
<th>AMD Opteron „Istanbul“</th>
<th>Intel Xeon „Westmere“</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 cores@2.8 GHz</td>
<td>6 cores@2.93 GHz</td>
</tr>
<tr>
<td>L1: 64 KB / L2: 512 KB / L3: 6 MB</td>
<td>L1: 32 KB; L2: 256 KB; L3: 12MB</td>
</tr>
<tr>
<td>2 X 2 X DDR2-800 (\rightarrow) 25.6 GB/s</td>
<td>2 X 3 X DDR3-1333 (\rightarrow) 63.6 GB/s</td>
</tr>
<tr>
<td>HT2000 (\rightarrow) 8 GB/s/dir</td>
<td>2 X QPI6.4 (\rightarrow) 12.8 GB/s/dir</td>
</tr>
</tbody>
</table>
Multicore-aware parallelization

Jacobi solver – a simple prototype

\[
\begin{align*}
d & \text{do } k = 1, N \\
& \quad \text{do } j = 1, N \\
& \quad \quad \text{do } i = 1, N \\
& \quad \quad \quad y(i,j,k) = b \ast (x(i-1,j,k) + x(i+1,j,k) + \\
& \quad \quad \quad x(i,j-1,k) + x(i,j+1,k) + \\
& \quad \quad \quad x(i,j,k-1) + x(i,j,k+1)) \\
& \quad \text{enddo} \\
& \text{enddo} \\
& \text{enddo}
\end{align*}
\]

■ A simple but representative test

■ Performance Metric: Million Lattice Site Updates per second (MLUPs)
  Equivalent MFLOPs: 6 FLOP/LUP * MLUPs

■ Bandwidth requirements: 16 Byte / Lattice Site Update (LUP) if write allocate on y is suppressed (24 Byte/LUP otherwise)

■ Performance estimate for best baseline: \(B_M / (16 \text{ Byte/LUP})\)
  \((B_M : \text{attainable memory bandwidth as measured with } \text{stream})\)
Multicore-aware parallelization
Towards multicore-awareness: Naive / classical approach

```c
!$OMP PARALLEL DO private(…)
do k=1,N
  do j=1,N
    do i=1,N
      y(i,j,k) = …
    enddo
  enddo
enddo
```

22. Juni 2010
SKALB – Multicore-aware parallelization
Multicore-aware parallelization

Reuse of in-cache data between threads: wavefronts

Reduce main memory accesses by a factor 2!

\( y(:, :, :) \) is obsolete!

Save main memory data transfers for \( y(:, :, :) \)!

Use ring buffer

\( \text{tmp}( :, :, 0:3) \)

which fits into the cache

Sync threads/cores after each \( k \)-iteration

\[
\begin{align*}
\text{core0: } & \quad x(:, :, \text{k-1:k+1})_t \\
& \Rightarrow \quad \text{tmp}( :, :, \mod(k, 4)) \\
\text{core1: } & \quad \text{tmp}( :, :, \mod(k-3, 4) : \mod(k-1, 4)) \\
& \Rightarrow \quad x(:, :, \text{k-2})_{t+2}
\end{align*}
\]
Running $\texttt{tb}$ wavefronts requires $\texttt{tb} - 1$ temporary arrays $\texttt{tmp}$ to be held in cache!

Max. performance gain: $\texttt{tb} = 4$

But extensive use of cache bandwidth!
Multicore-aware parallelization

Wavefront – Jacobi on state-of-the-art multicores

Compare against optimal baseline!

Performance gain $\sim B_{olc} = \text{L3 bandwidth} / \text{memory bandwidth}$

22. Juni 2010

SKALB – Multicore-aware parallelization
Multicore-aware programming
First lectures on the node level

- Fast shared on-chip resources: Room for new ideas!
  - Fast thread synchronisation through shared caches
    - Wavefront works fine with Intel compilers but fails with gcc:
      OMP BARRIER: 4-core Core i7:
        - 11,000 cycles (gcc 4.3.3) vs. 800 cycles (icc 11.0)

  → Ability to exploit parallel structures at finer granularity
    (shorter loops, frequent synchronisation)

- Technique can easily be extended to many regular stencil based
  iterative methods, e.g.
  - Gauß-Seidel (→ done)
  - Lattice-Boltzmann flow solvers (→ work in progress)
  - Multigrid (→ BMBF proposal EMMMA)

- For multi-node applications an hybrid MPI+OpenMP approach
  needs to be implemented (next slides)
Temporal blocking requires multi-layer halo

Using \textit{diagonal communication elimination} (DCE) (Ding/He SC 2001)

Exchanging halo with neighbors done only along the coordinate directions

More complex stencils, e.g. occurring at lattice Boltzmann methods, need more attention for deciding which data to communicate
Multicore-aware programming
Impact of Multi-layer Halo on Performance

- **Assumptions for model:**
  - No overlap between communication and computation
  - QDR-InfiniBand
    - 3.2 GB/s
    - 1.8 µs latency
  - Node performance
    - 2 GLUP/s

- Reduced latency by message aggregation
- Degrade due to halo work
- No impact for large domain sizes

**Impact of Multi-layer Halo on Performance**

- Reduced latency by message aggregation
- Degrade due to halo work
- No impact for large domain sizes

**strong scaling**
Multicore-aware programming

Strong Scaling

- Nehalem EP cluster
- QDR Infiniband

- Domain size: $600^3 \approx 1.6$ GB

- Benefit of temporal blocking eaten by communication overhead ($\sim 60\%-70\%$)

- Pipelining 1PPN suffers from bulk-synchronized communication
Multicore-aware programming

Weak Scaling

- Nehalem EP Cluster
- QDR Infiniband
- Subdomain size per node: $600^3$
- Benefit from temporal blocking can be maintained
Parallel GPU computing with Lattice-Boltzmann solvers— a focus of SKALB
### Preliminary comparison Blue-Gene/P – GPU-Cluster

**LUDWIG@iRMB-TuB**
- 96 GPUs: NVIDIA Tesla
- 24 host nodes
- D3Q13 discretization

<table>
<thead>
<tr>
<th></th>
<th>#cores</th>
<th>precision</th>
<th># grid nodes</th>
<th>absolute performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue-Gene/P</td>
<td>16.384</td>
<td>double</td>
<td>$10^9$</td>
<td>~20 GLUPS</td>
</tr>
<tr>
<td>Ludwig (96 GPUs)</td>
<td>23.040</td>
<td>single</td>
<td>$3.2 \times 10^9$</td>
<td>~45 GLUPS</td>
</tr>
</tbody>
</table>
### WALBERLA:
*Block-/patch-structured heterogeneous CPU/GPU*

**D3Q19**

- Halo exchange between blocks
- Single precision computations

#### GPU cluster at HLRS

<table>
<thead>
<tr>
<th>Blocks</th>
<th>GPU: 1</th>
<th>GPU: 22, CPU:1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>1</td>
<td>30 1 30 60 90</td>
</tr>
<tr>
<td>Processes</td>
<td>2 x GPU</td>
<td>60 x GPU 2 x GPU + 60 x GPU + 60 x GPU +</td>
</tr>
<tr>
<td></td>
<td>6 x CPU</td>
<td>180 x CPU 420 x CPU 660 x CPU</td>
</tr>
<tr>
<td>MFLUPS</td>
<td>476</td>
<td>14480 459 13267 15684 17846</td>
</tr>
</tbody>
</table>

~3 TFlop/s

22. Juni 2010

SKALB – Multicore-aware parallelization
Besten Dank

- Financial support through KONWIHR-II projects OMI4papps, HQS@HPC

BMBF Project
SKALB (01 IH08003A)
www.skalb.de

22. Juni 2010
http://www.rrze.uni-erlangen.de/hpc/