Masterpraktikum Scientific Computing – High Performance Computing

SWE Project Presentation and Results

Gaurav Kukreja
Evangelos Drossos

21.01.2014

Optimizations Performed

- Instrumentation with Scalasca
- Vectorization with Intel Cilk
- OpenMP optimization
- MPI optimization

Instrumentation

- Instrumented code to get a better idea of bottleneck issues and points of improvement
- Analyzed performance after each optimization

Instrumentation

- Initial Analysis
  - Showed that most significant time was spent in *computeNumericalFluxes()*
  - Waiting time for MPI Communication was significant

Vectorization

- Vectorization using Intel Cilk
  - Better Readable Code
  - Trust the compiler
- Split computation of rows into chunks. Computed chunks in one iteration of *computeNetUpdates()*
- After testing we determined an optimal chunk length of 8.

Vectorization

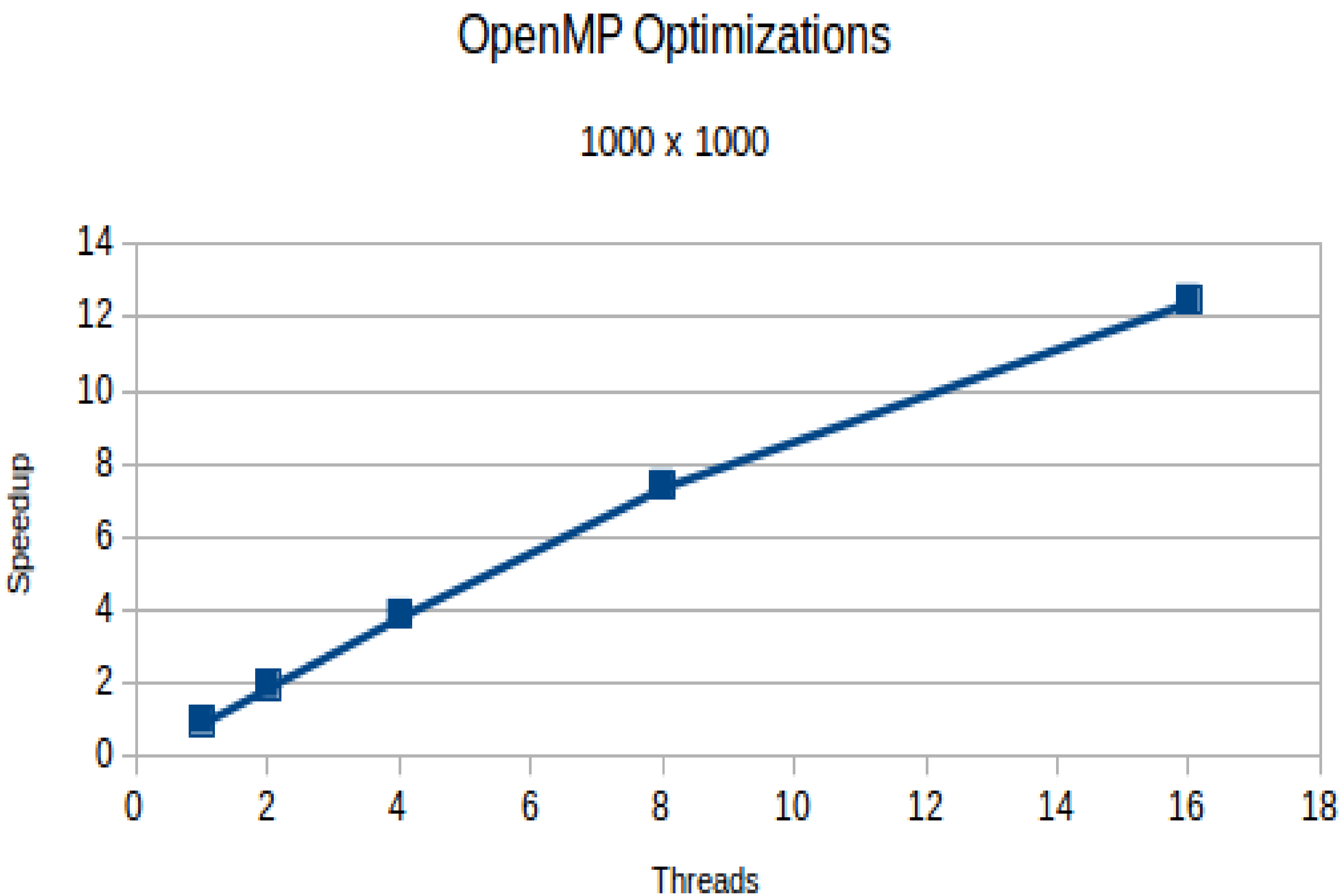
- Though we expected a large improvement, we achieved only slight improvement.
  - The compiler was already vectorizing the code.
  - Performance gets worse for larger block sizes. For block size 1, nearly same performance.
  - Based on tests on login node, performance ~7% better for chunk length 8

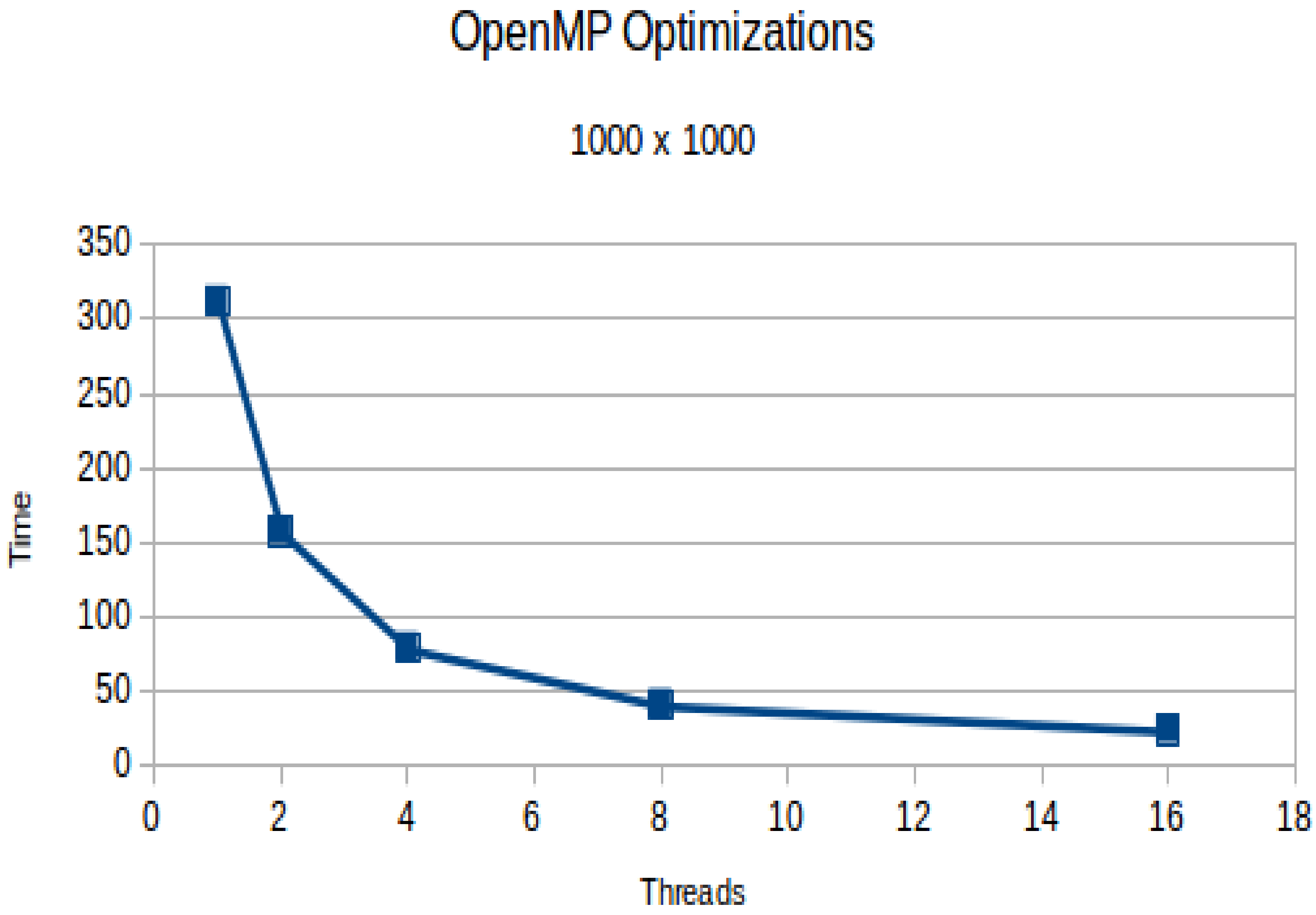
OpenMP

- Existing code is parallelized using OpenMP.
- Fixed a few compilation issues, and minor modifications
  - schedule(static) nowait
- Fused loops performing the calculations for the vertical and horizontal edge updates
- No cache optimization performed.

OpenMP

- Slight Improvement in performance.
  - ~1.5% against existing code.
  - Near Linear Speedup until 8 threads.
  - For more threads, slack expected.


OpenMP Optimizations
1000 x 1000
Speedup
Threads
0
2
4
6
8
10
12
14
16
18



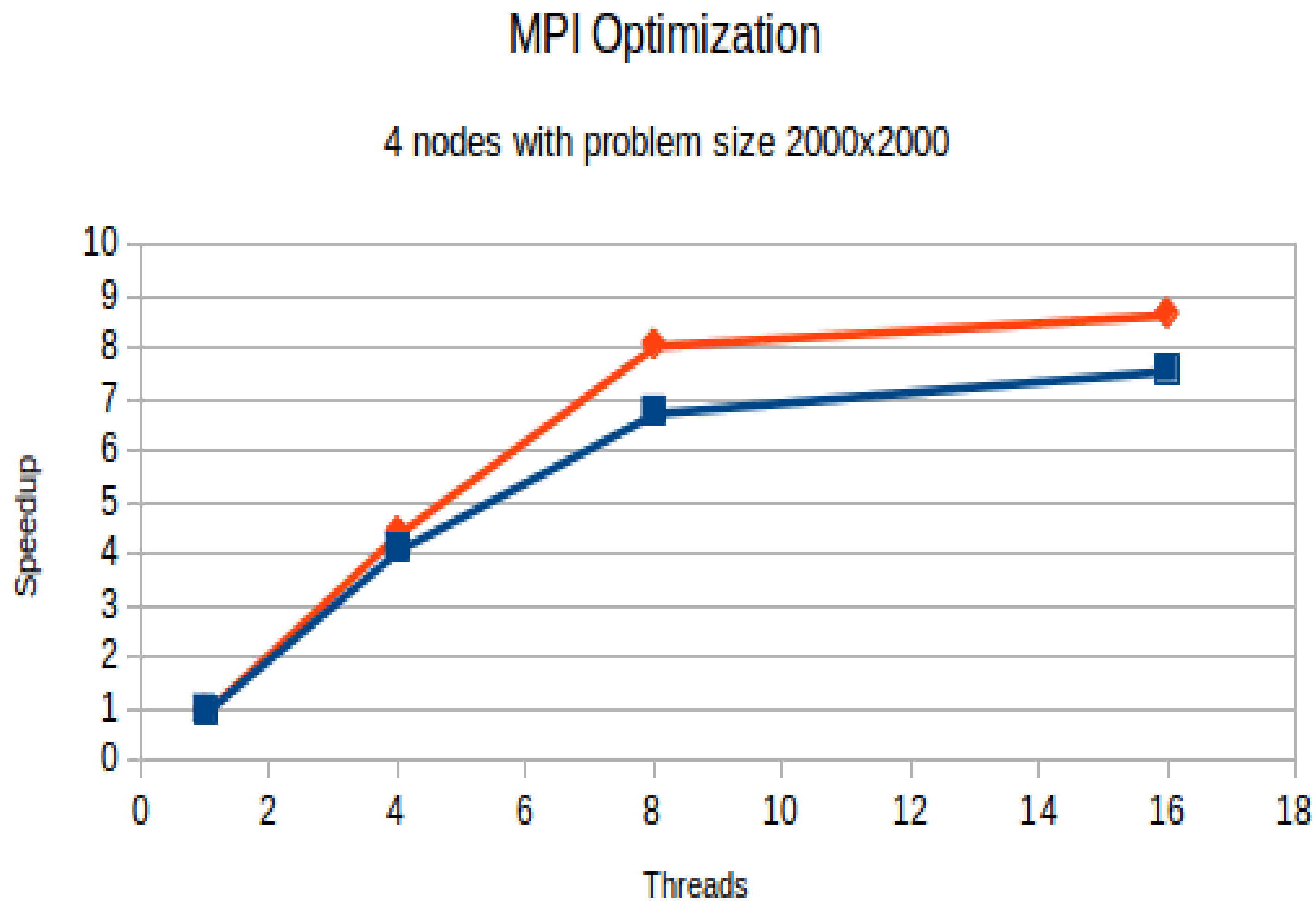
OpenMP Optimizations
1000 x 1000
Time
350
300
250
200
150
100
50
0
0
2
4
6
8
10
12
14
16
18
Threads

Hybrid Approach

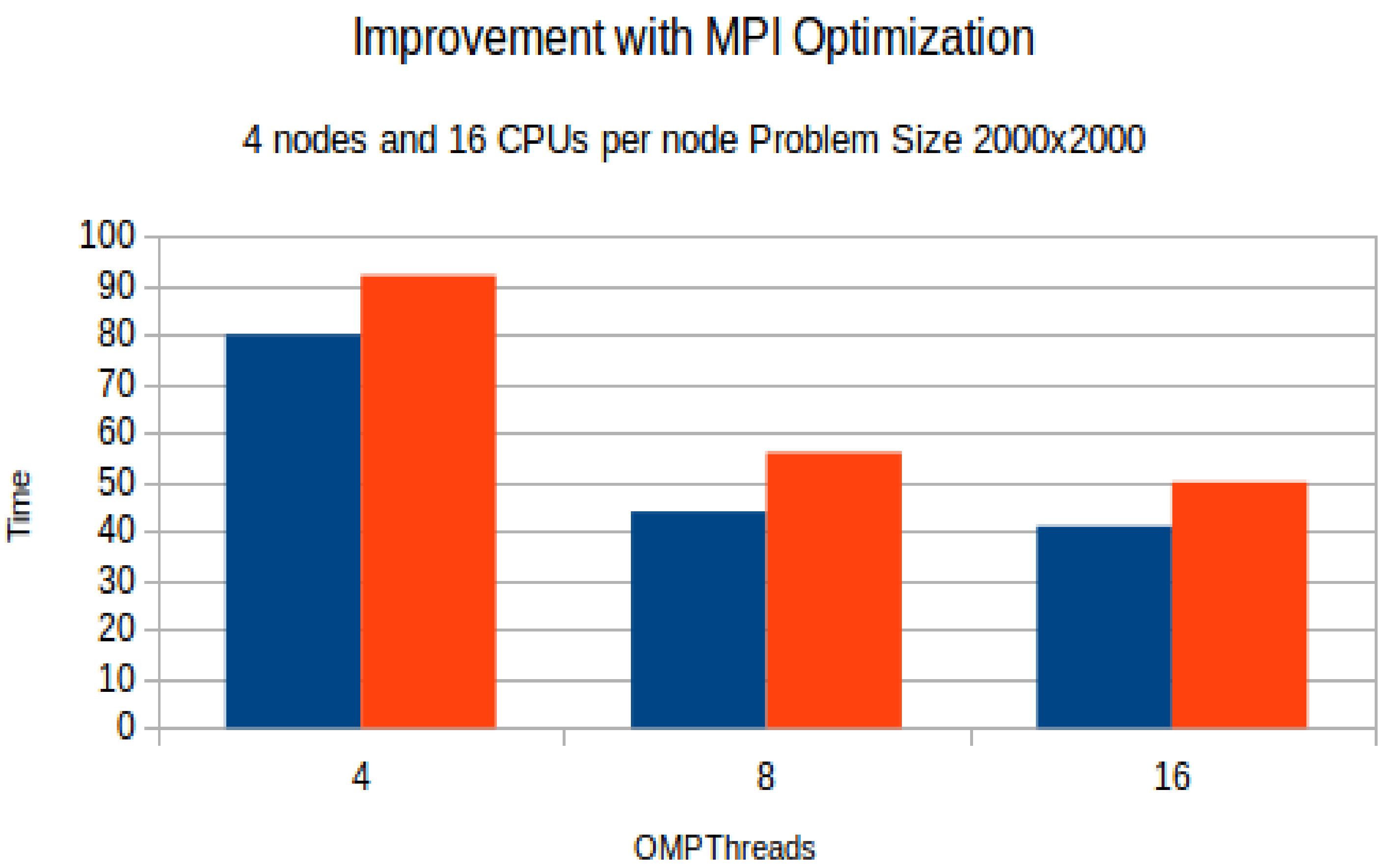
- Benefits of using MPI with OpenMP
  - OpenMP uses shared memory, which is good for cores on one node which share L2 Cache.
  - MPI provides flexibility and control for sending data between threads.
  - Hybrid approach can run optimally on mutliple nodes.
  - Each MPI Thread runs on a node, and creates OpenMP Threads to run on the cores.

MPI

- Existing MPI Code used Blocking MPI_Sendrecv()
- To make Communication asynchronous and overlap with computation
  - Used non-blocking MPI_Isend() and MPI_Irecv()
  - Called computeNumericalFluxes_innerBlock() to compute horizontal and vertical edge updates for inner block
  - Wait and ensure communication was complete.
  - Called computeNumericalFluxes_borders().
- ~12% improvement in performance



MPI Optimization
4 nodes with problem size 2000x2000
Speedup
Threads
0
1
2
3
4
5
6
7
8
9
10
0
2
4
6
8
10
12
14
16
18



Improvement with MPI Optimization
4 nodes and 16 CPUs per node Problem Size 2000x2000
Time
100
90
80
70
60
50
40
30
20
10
0
4
8
16
OMPThreads

Hybrid Approach

- Worth noting
  - Hybrid code with problem size 2k x 2k on 4 nodes, and 8 CPUs takes slightly more time as OpenMP code with problem size 1k x 1k on 8 CPUs
  - The time difference is due to communication.
  - Overall improvement in performance ~10%