Topology-aware OpenMP Process Scheduling

Peter Thoman, Hans Moritsch, and Thomas Fahringer
University of Innsbruck (Austria)

Motivation

Motivation – Hardware Trends

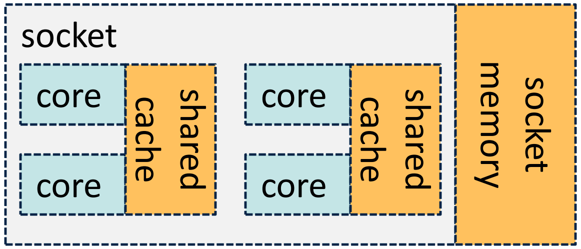
- Multi-core, multi-socket NUMA machines are in wide use in HPC
  - Complex memory hierarchy and topology
  - Large number of cores in single shared memory system

→ are existing OpenMP applications and implementations ready?

Motivation – Hardware Trends

- Multi-core, multi-socket NUMA machines are in wide use in HPC
  - Complex memory hierarchy and topology
  - Large number of cores in single shared memory system

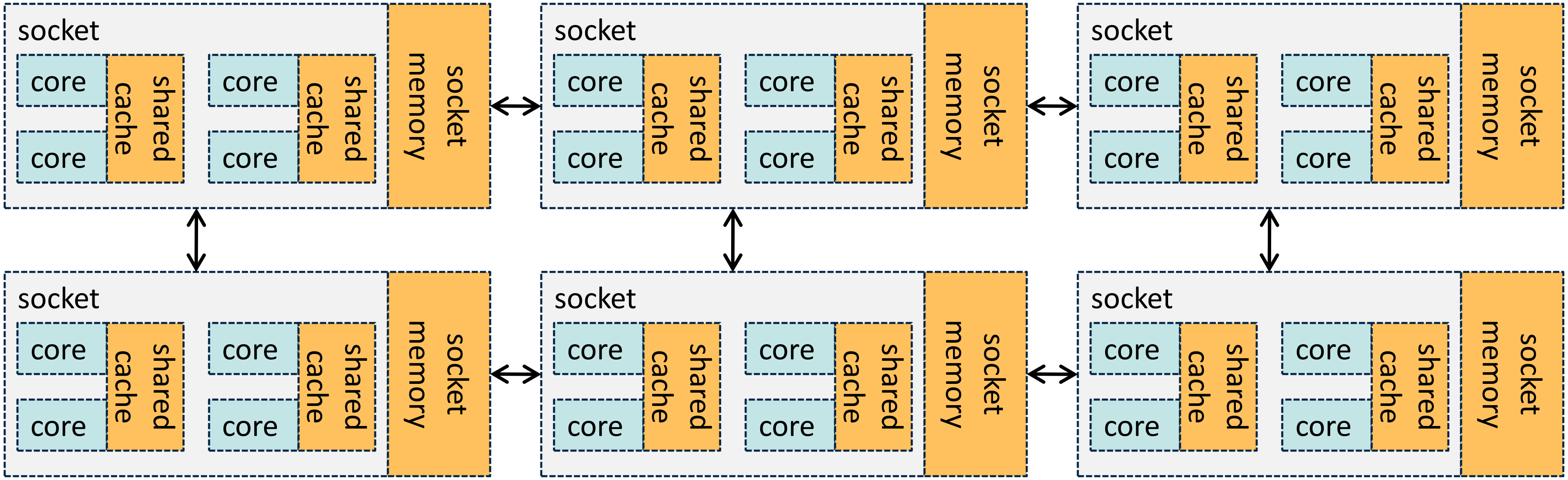
→ are existing OpenMP applications and implementations ready?



Motivation – Hardware Trends

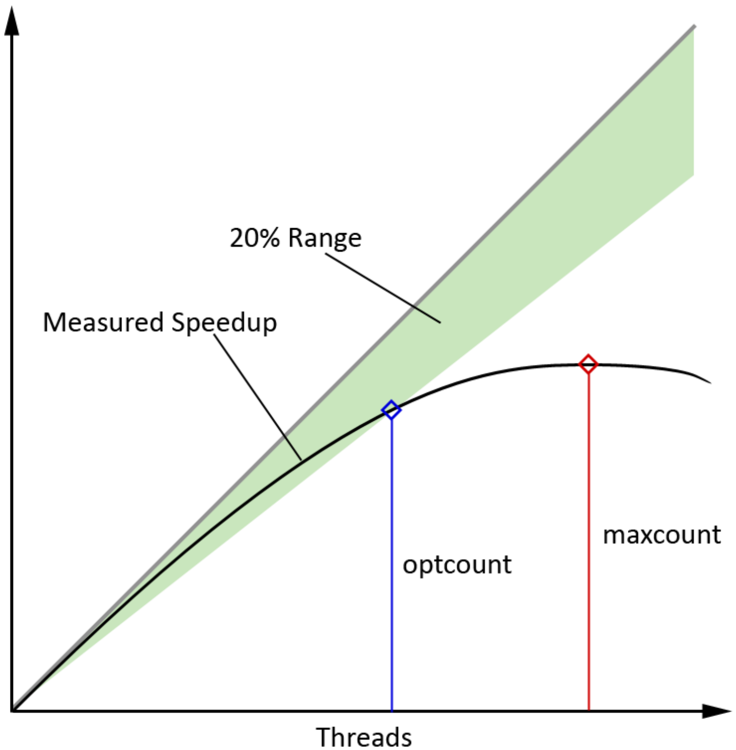
- Multi-core, multi-socket NUMA machines are in wide use in HPC
  - Complex memory hierarchy and topology
  - Large number of cores in single shared memory system

  → are existing OpenMP applications and implementations ready?



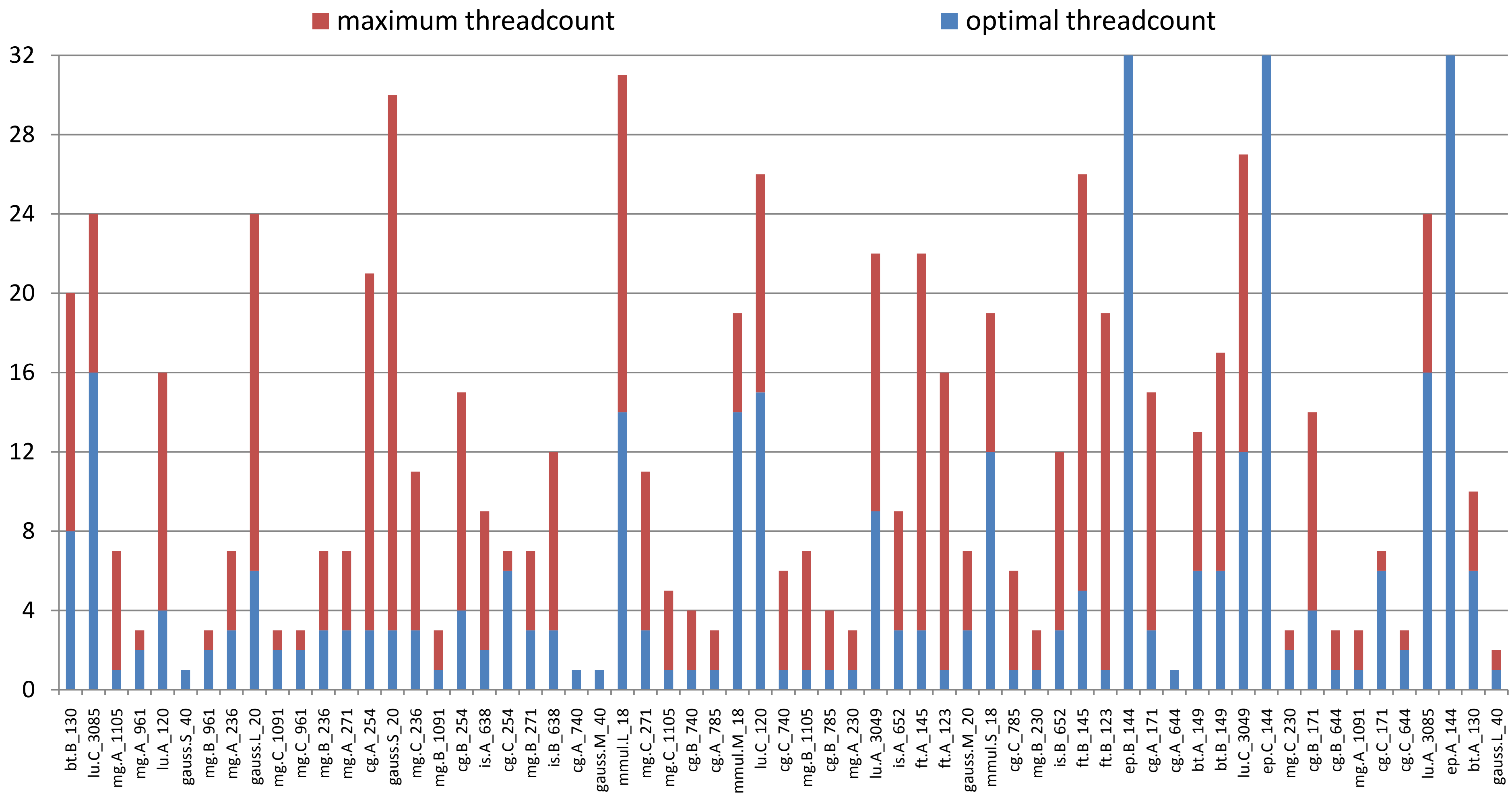
Scalability

- We profiled individual OpenMP parallel regions in a variety of programs and problem sizes
- On a 8-socket quadcore NUMA system (32 cores)
- Determine two metrics:
  - Maximum threadcount
    - Maximum amount of threads that can be used with some speedup
  - Optimal threadcount
    - Maximum amount of threads that allows a speedup within 20% of ideal



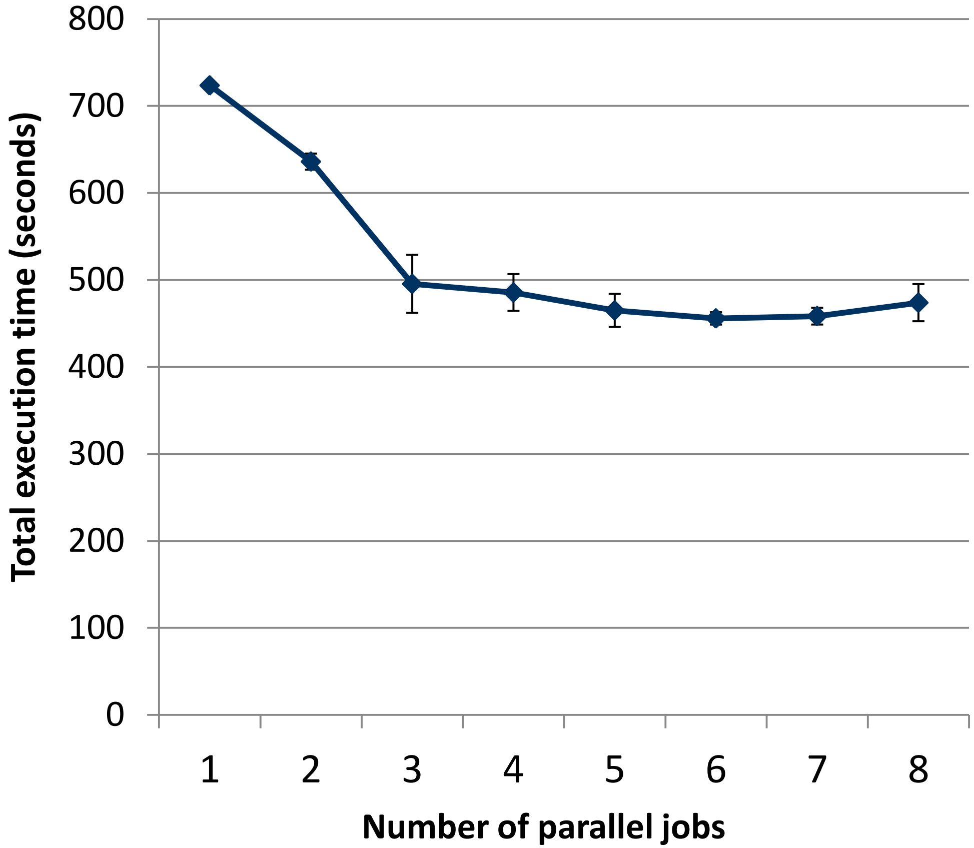
Scalability Results

informatik
institut für
in
insieme
compiler project
universität
innsbruck



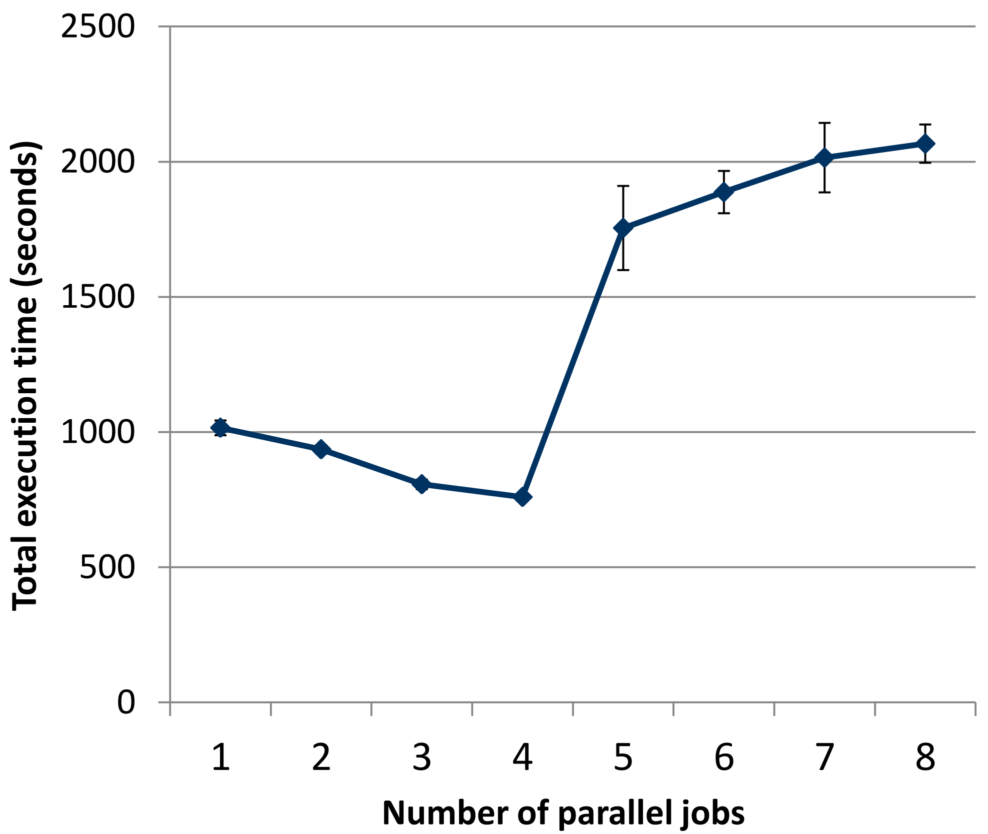
Motivation – Multi-Process

- First idea: run more than one OMP program (job) in parallel



Motivation – Multi-Process

- Of course it is not always that simple – a different workload:



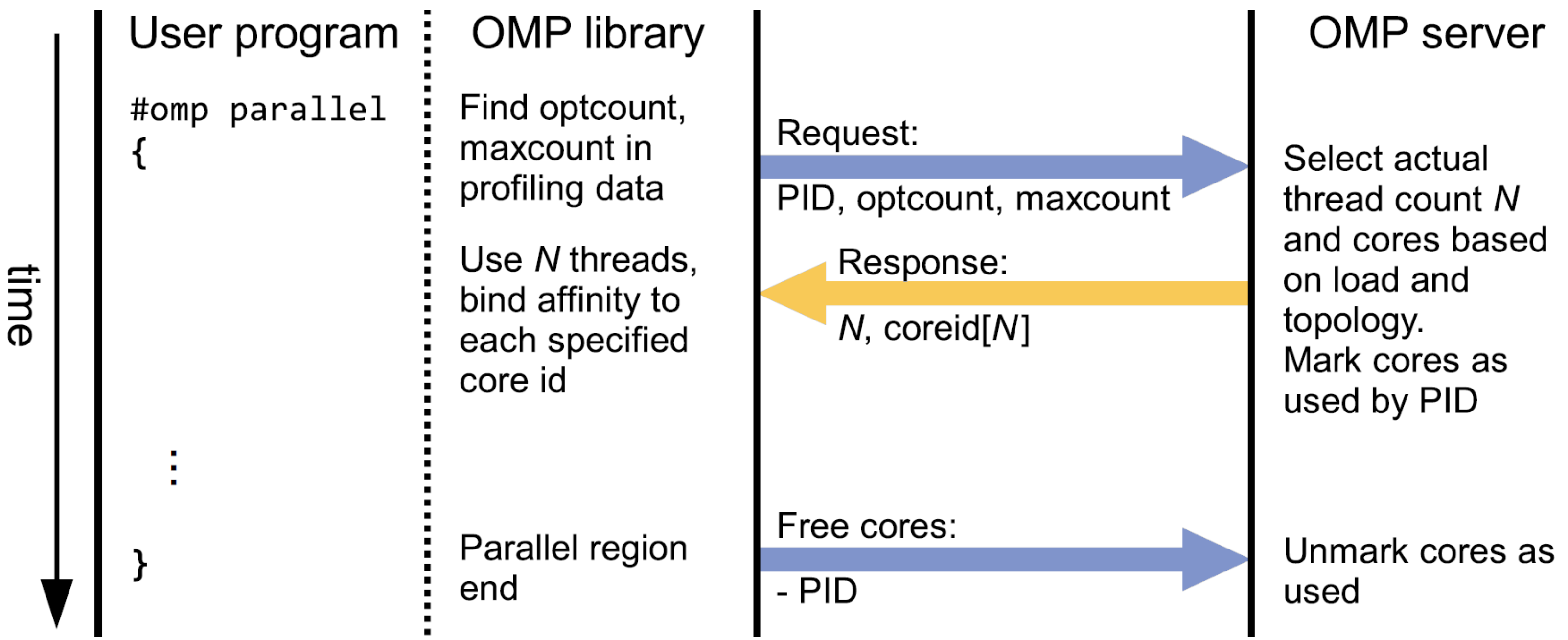
Algorithm & Implementation

Multi-Process Scheduling Architecture

- Goal:
  - Facilitate system-wide scheduling of OpenMP programs
- Basic Design:
  - One central control process (*server*), message exchange between server and the OMP runtime of each program
- Message protocol:
  - Upon encountering a OMP parallel region:
    - OMP processes send a request to server for resources
      - Includes scalability information for region
    - Use cores indicated by reply
  - When leaving region send signal to free cores

Implementation & Flow

- Based on UNIX message queues
  - Well suited semantically and fast enough (less than 4 microseconds roundtrip on our systems)



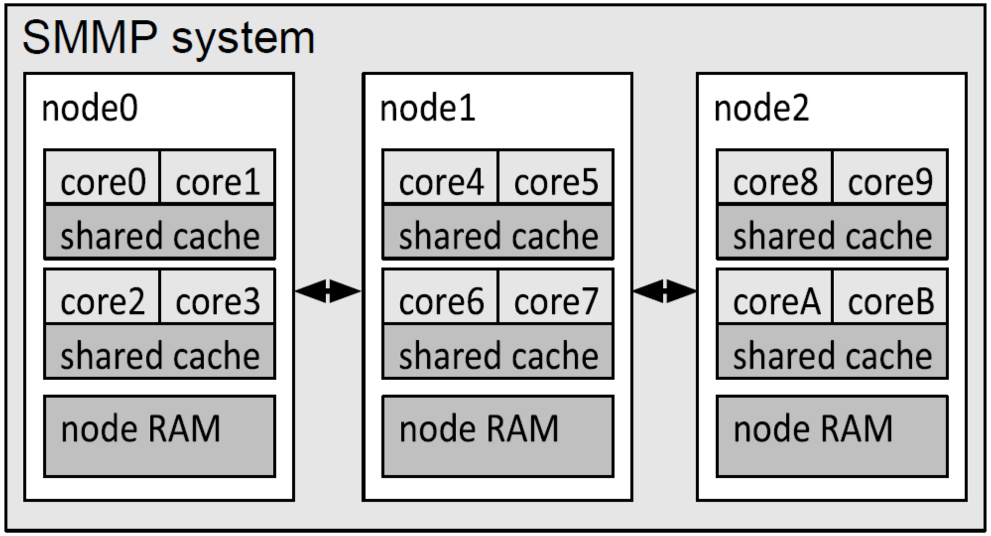
Topology-aware Scheduling Algorithm

- Multi-process scheduling ameliorates many-core scalability problems
- What about complex memory hierarchy?
  - Make server topology aware
  - Base scheduling decisions on
    - Region scalability
    - Current system-wide load
    - System topology

➔ Topology-aware OMP scheduler

Topology Representation

- Distance matrix for all cores in a system
  - Higher distance amplification factors for higher levels in the memory hierarchy
- Example:



	0	1	2	3	4	5	6	7	8	9	A	B
0	0	0	1	1	10	10	10	10	20	20	20	20
1	0	0	1	1	10	10	10	10	20	20	20	20
2	1	1	0	0	10	10	10	10	20	20	20	20
3	1	1	0	0	10	10	10	10	20	20	20	20
4	10	10	10	10	0	0	1	1	10	10	10	10
5	10	10	10	10	0	0	1	1	10	10	10	10
6	10	10	10	10	1	1	0	0	10	10	10	10
7	10	10	10	10	1	1	0	0	10	10	10	10
8	20	20	20	20	10	10	10	10	0	0	1	1
9	20	20	20	20	10	10	10	10	0	0	1	1
A	20	20	20	20	10	10	10	10	1	1	0	0
B	20	20	20	20	10	10	10	10	1	1	0	0

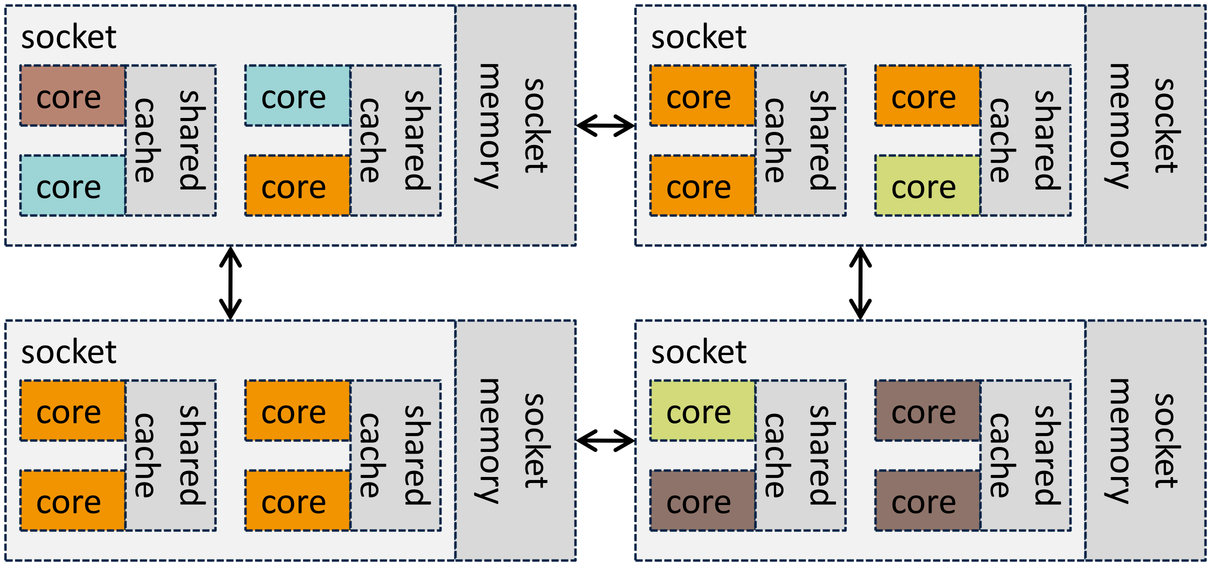
Simple Scheduling

- Request from region with given `maxcount` and `optcount`:
  1. N = optcount + loadfactor * (maxcount - optcount)
     - loadfactor dependent on amount of free cores
  2. Select N-1 cores close to core from which the request originated

- Slightly more complicated in practice
  - dealing with case where fewer than N cores available (decide whether to queue or return smaller amount)

Fragmentation

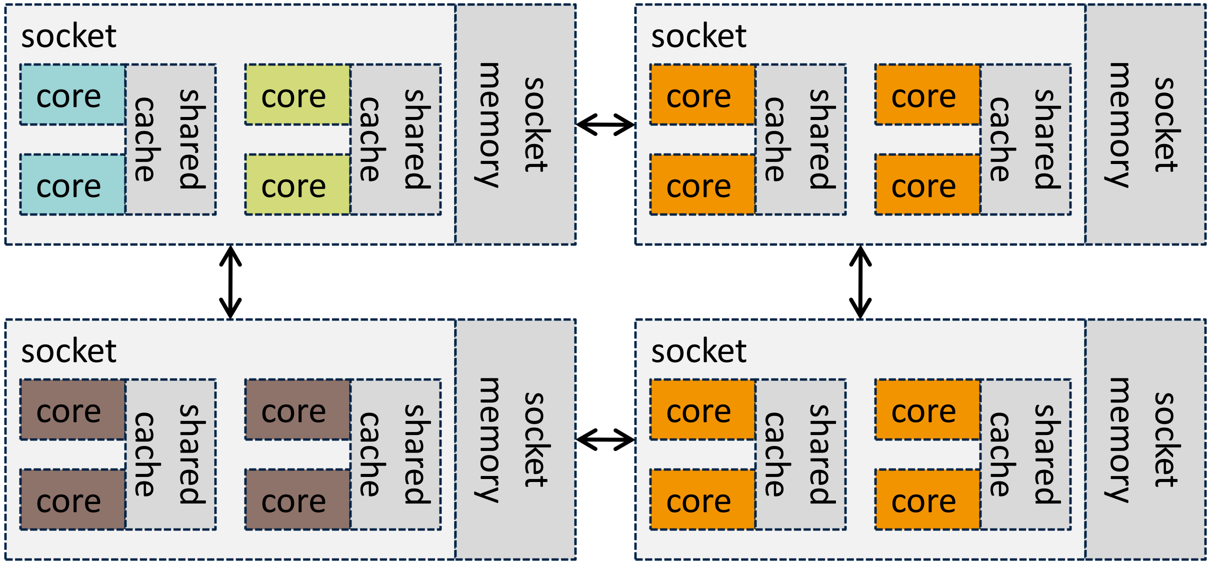
- Using simple scheduling leads to *fragmentation*:



- Sum of local distance in all 4 processes: 44

Improvement: Clustering

- Same processes without fragmentation:



- Sum of local distance in all 4 processes: 13

Clustering Algorithm

- Moving threads once started has significant performance impact (caches, pages, etc)
  → instead change algorithm to discourage fragmentation

- Define cores as part of a hierarchy of *core sets*
- When selecting a core from a new set, prefer (in order)
  1. A core set containing exactly as many free cores as required
  2. A core set containing more free cores than required
  3. An empty core set

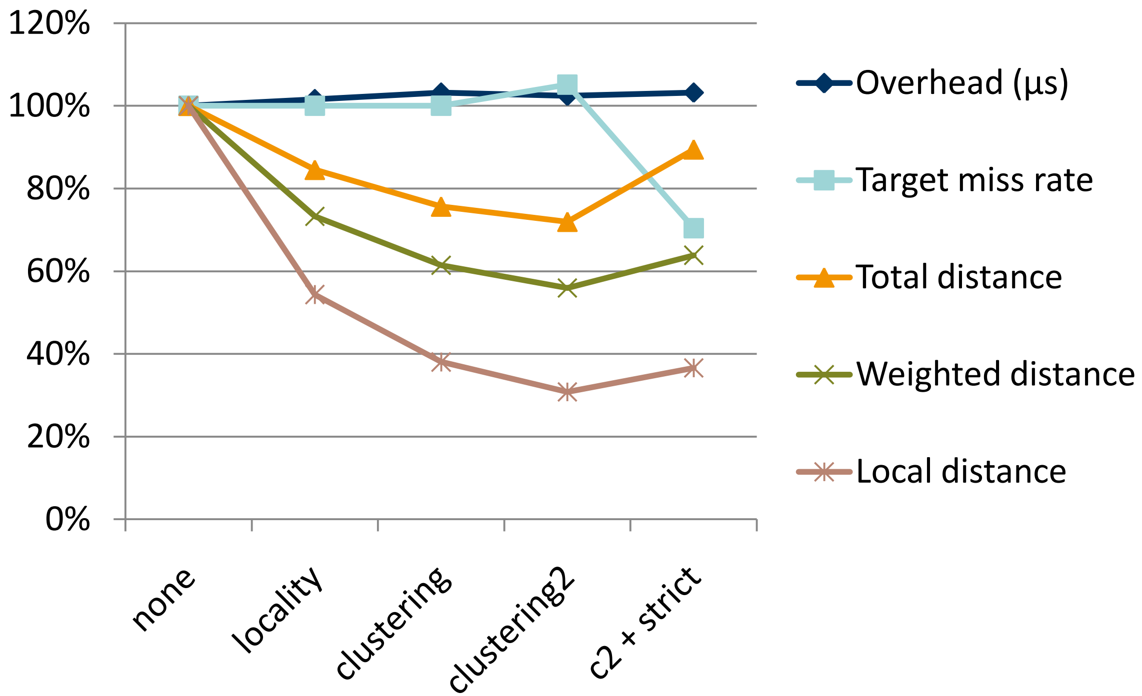
- Further improvement possible by adjusting number of selected cores (*enhanced clustering*)

Evaluation

Simulation

- Evaluate impact of scheduling enhancements over 10000 semi-random requests
- Calculate or measure 5 properties:
  - Scheduling time required per request
  - Target miss rate: |#returned_threads - #ideal_threads|
  - 3 distance metrics:
    - Total distance: from each thread in a team to each other
    - Weighted distance: distance between threads with close id weighted higher
    - Local distance: only count distance from each core to next in sequence

Simulation Results



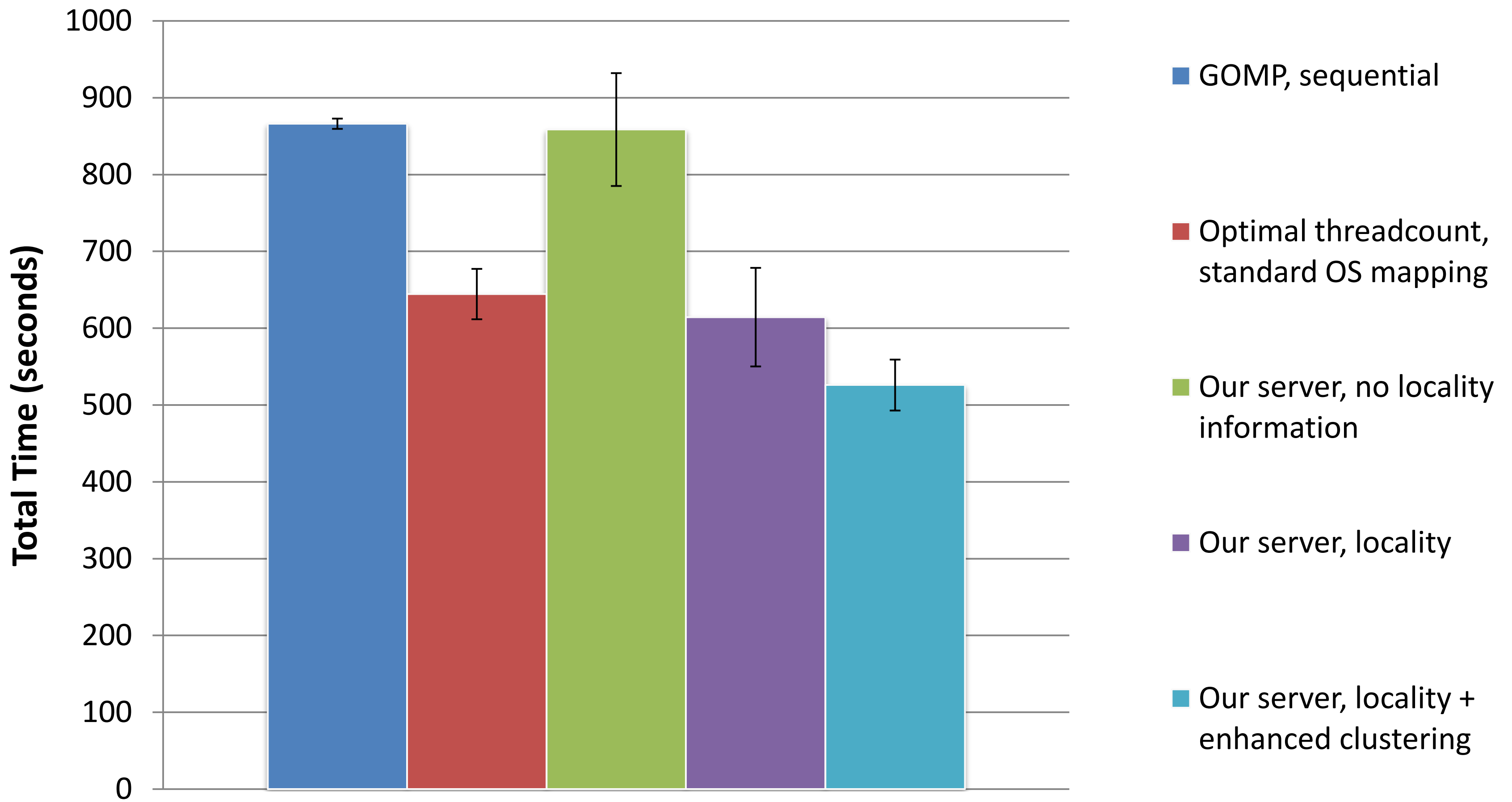
- Absolute overhead always below 1.4 microseconds
- Enhanced clustering reduces local distance by 70%

Experiments

- Hardware:
  - Sun XFire 4600 M2
  - 8 quad-cores (AMD Barcelona, partially connected, 1-3 hops)
- Software
  - Backend: GCC 4.4.2
  - “Default” OMP: GOMP
  - Insieme compiler/runtime r278

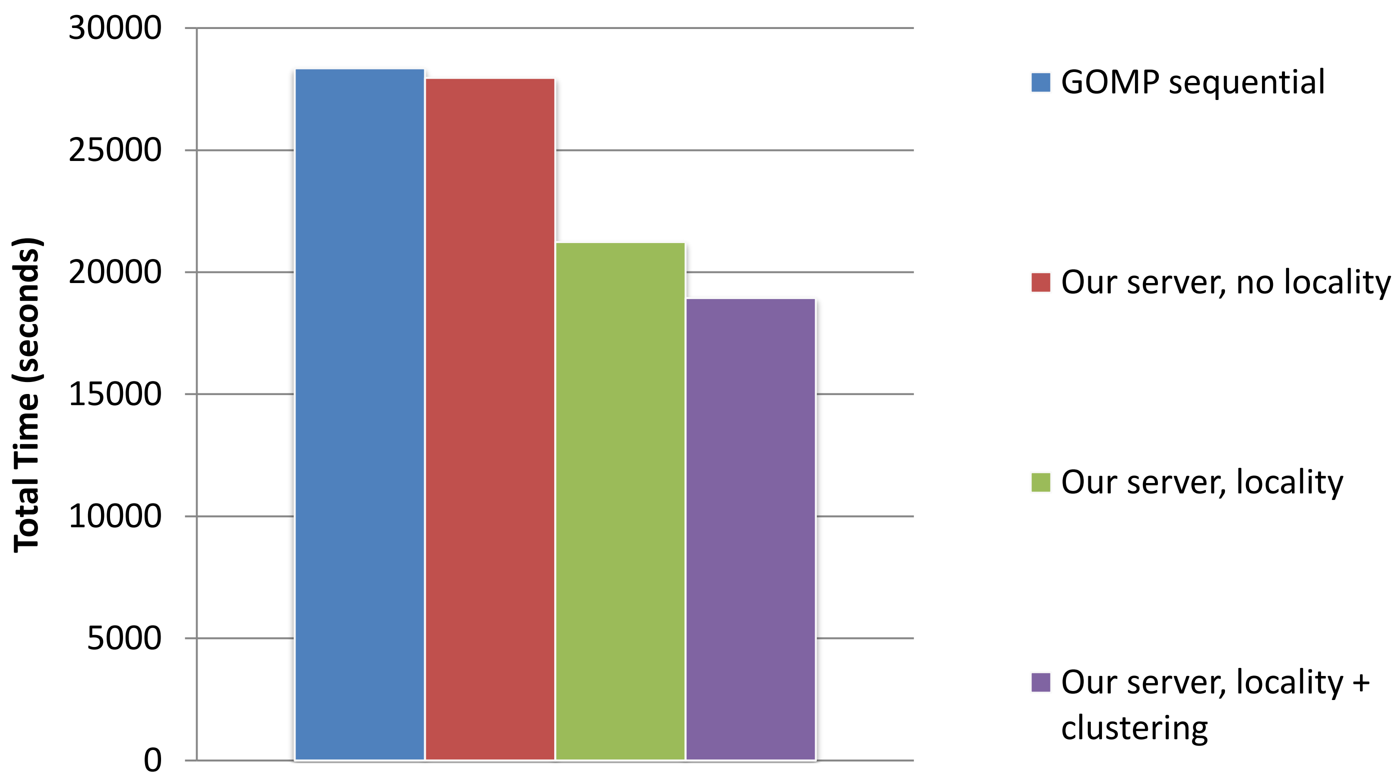
Small-scale Experiment

- Random set of 13 programs tested



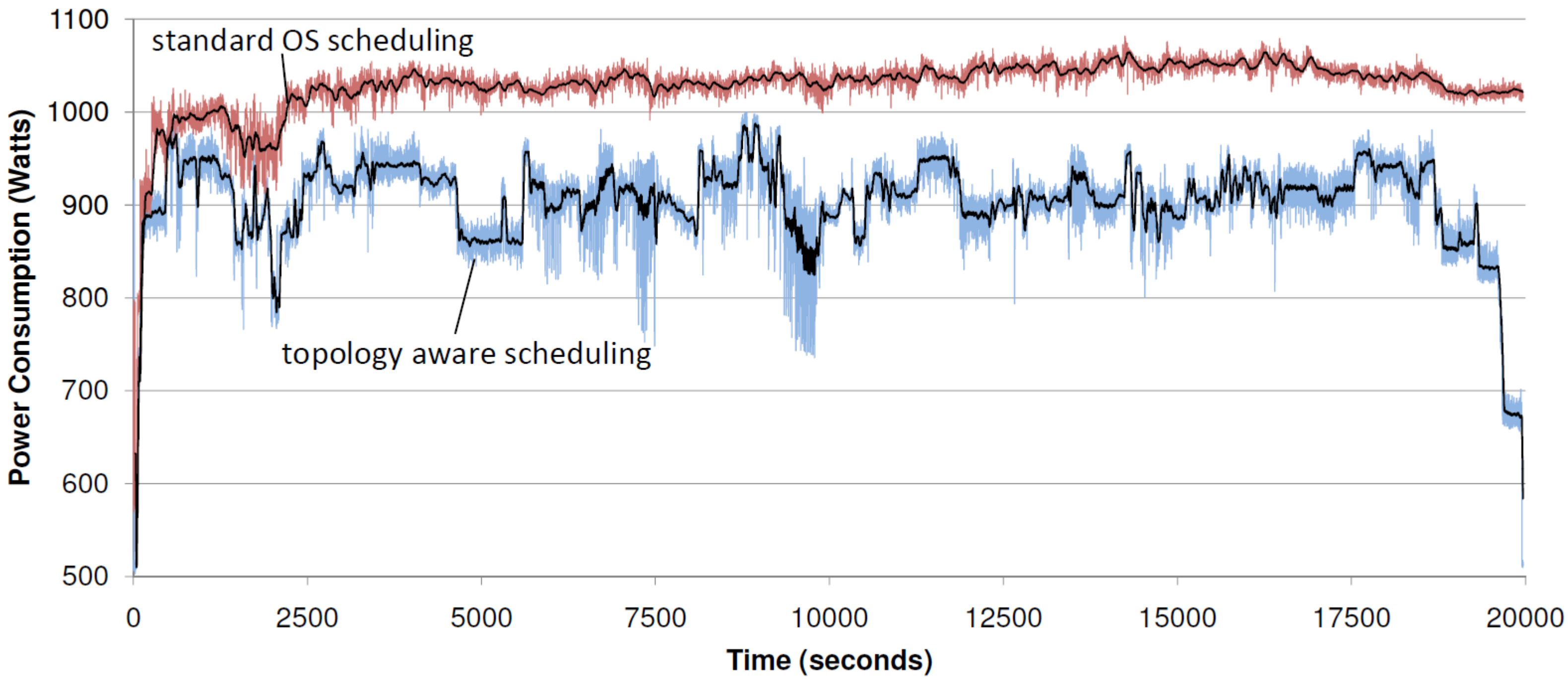
Large-scale Experiment

- Random programs chosen from NPB & 2 kernels
- Random problem sizes



Power Consumption

- Power consumption measured during large-scale experiment:



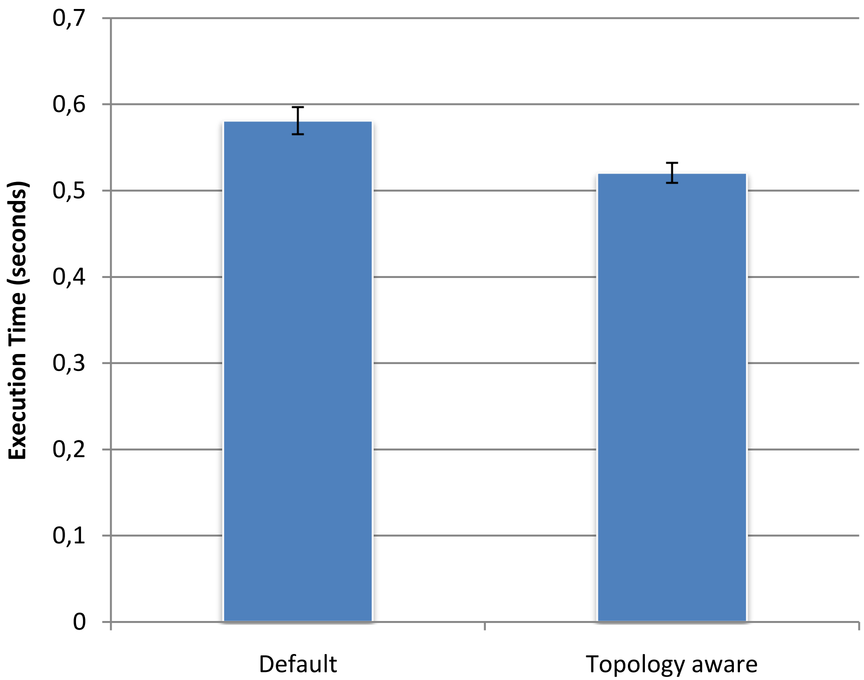
- Topology-aware scheduling (with appropriate thread counts) reduces average power consumption

Hybrid MPI/OpenMP

- One program consists of more than one process
- Our topology-aware thread mapping meaningful even for a single program in this case

- Test of an ADI solver, 8 MPI processes and 4 threads each
- Improvement is around 11%

- OpenMPI used in both cases



Summary and Conclusion

Summary

- Central OpenMP server process
  1. Selects number of threads for parallel regions depending on
     - Scalability information
     - System load
     - Clustering considerations
  2. Performs topology-aware mapping of threads to cores

- Evaluation
  - Up to 33% performance improvement compared to standard scheduling
  - Additional reduction in power consumption

Future Work

- How to determine/estimate region scalability without exhaustive profiling
- Make external non-OMP load impact scheduling decisions

Thank you!