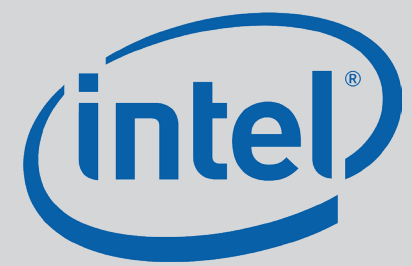


# Engineering Multi-Threaded Codes for Hierarchical Memory Architectures



European Research and Innovation Conference 2010, Braunschweig, Germany

C. Terboven, D. Schmidl, P. Kapinos and D. an Mey  
Center for Computing and Communication, RWTH Aachen University  
{terboven, schmidl, kapinos, anmey}@rz.rwth-aachen.de

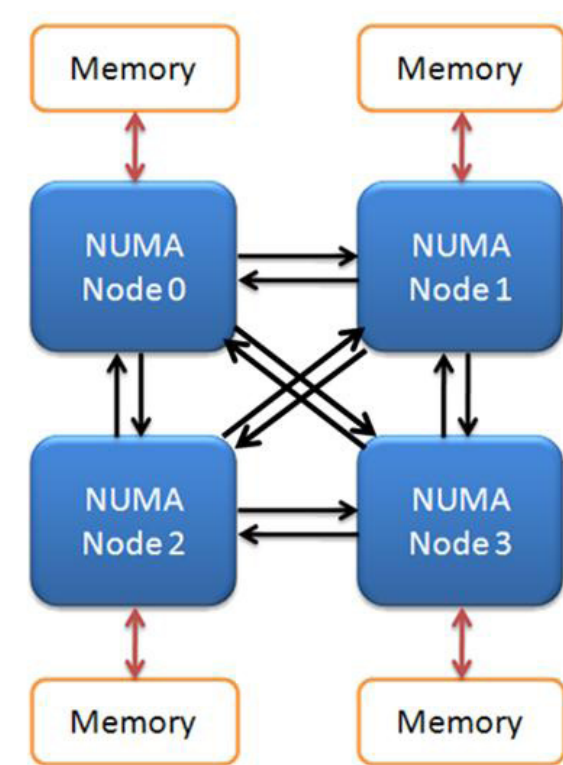
## cc-NUMA Architectures

Still many programmers are taken by surprise when ignoring performance implications of modern cc-NUMA architectures.

- What is a cc-NUMA architecture?
- What are the performance implications of cc-NUMA architectures?
- What are the standard tools to deal with performance implications?
- What else can be done?

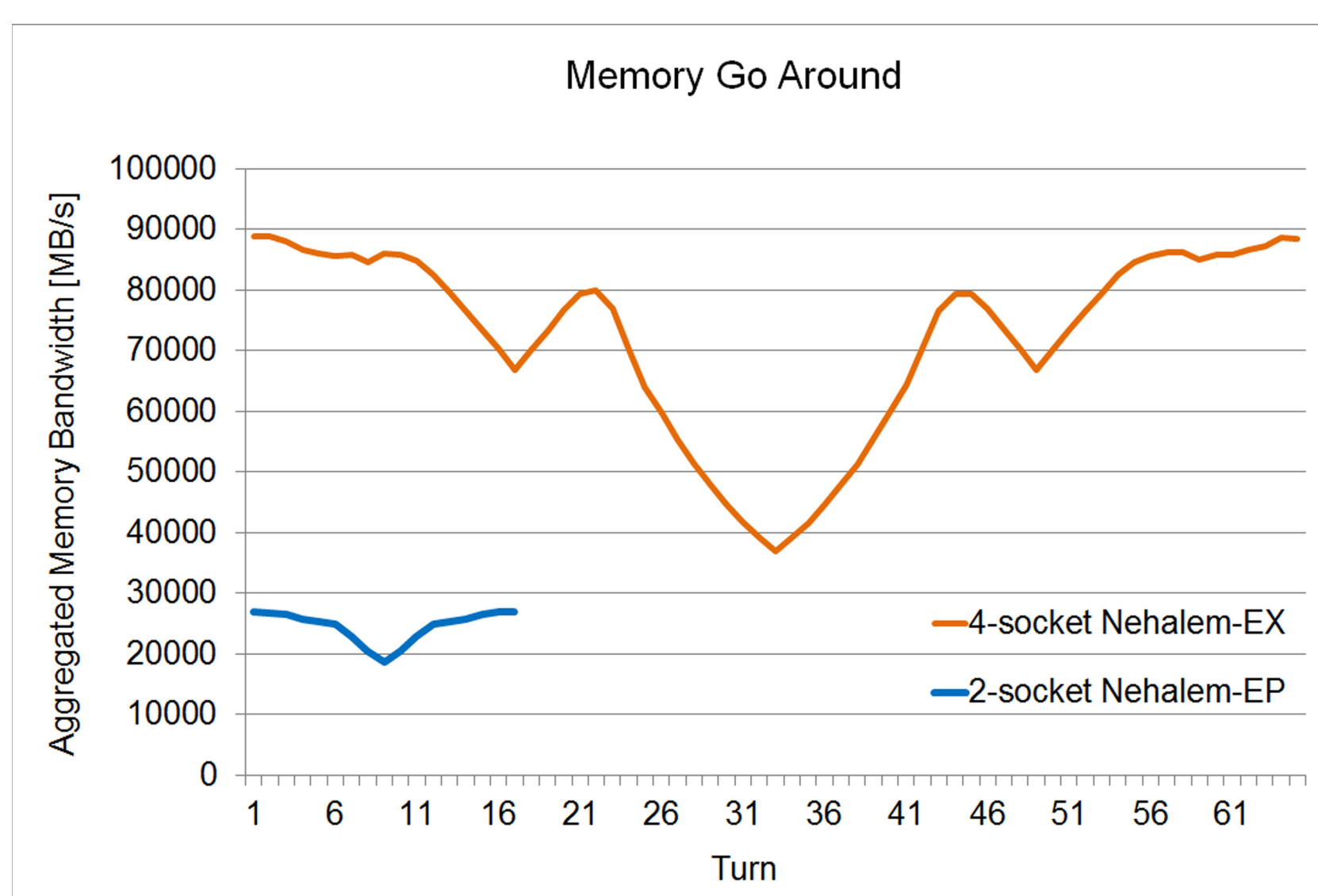
### cc-NUMA Architecture – here: 4-socket Nehalem-EX

- Processors equipped with local memory and connected with each other through cache-coherent interconnect
- Therefore: **fast local** access but **slower remote** access
- Careful data placement is very important
- Cache-friendly codes may not suffer that much
- HPC codes frequently do.



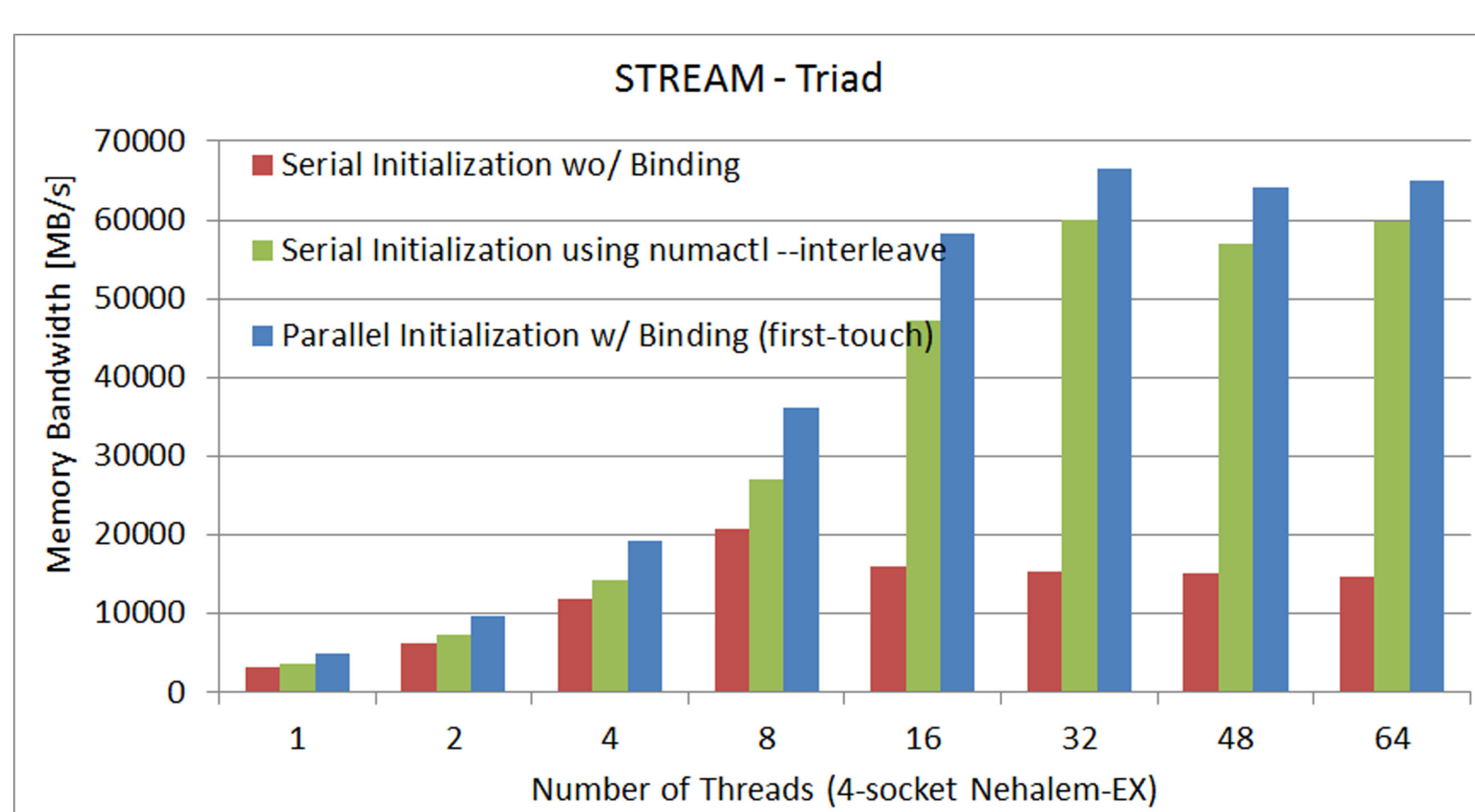
### Performance Implications - here: Memory-Go-Around benchmark:

- Memory allocated by and close to core number  $i$  is accessed by core number  $(i+\text{turn}) \bmod n$  for  $i=0\dots n-1$ , aggregated over all cores



### Tools for dealing with cc-NUMA effects

- By default the first-touch memory allocation policy of the operating system (OS) allocates a data page close to the thread which touches it
- **Binding threads** keeps the OS from moving threads away from its data
- If the data access pattern is not known a-priori or may change over time:
  - **Round Robin Memory Placement** via numactl tool may be beneficial
  - **Page migration** can be employed to explicitly move data to threads, if supported by the OS
  - **Next-touch migration**: Pages are moved to the thread that accesses them next, if supported by the OS
  - **Automatic migration** may considerably increase overhead
- No popular productive solution yet



## Object-Orientation

Object-oriented programming can not only be made work well with OpenMP, but it can also be used to hide the complexity of numa-aware memory management, i.e. in our laperf library.

```
vector<double, OpenMPInternalPar> x, y; matrix_crs<double,
MemoryPolicy::Chunked> A;
```

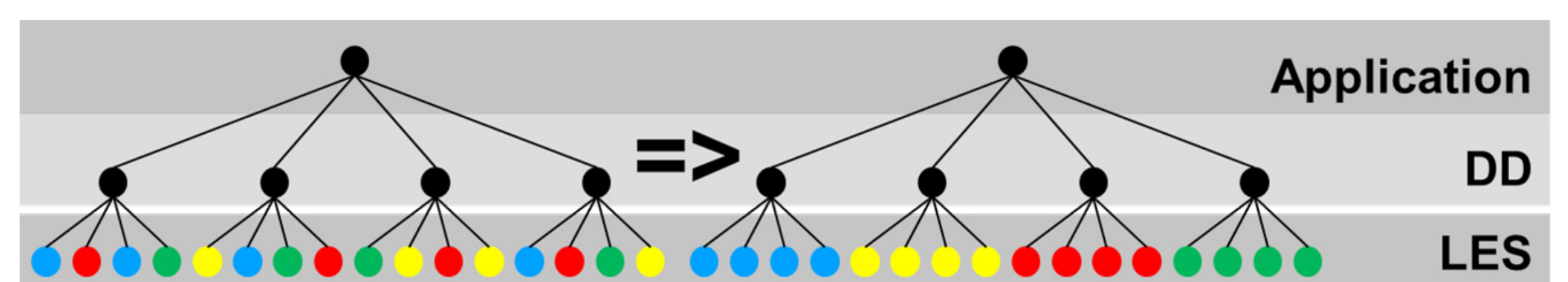
$$y = A * x;$$

Parallel (sparse) matrix vector multiplication  
Matrix multiplication using chunked distribution strategy

- laperf uses **Expression Templates** to translate an expression into corresponding template expression resembling the AST:
  - Compiler unrolls this expression into single loop over vector; this loop can be parallelized efficiently
- cc-NUMA aware memory allocation can be carried out by the **allocator** (STL data type abstracting memory management) via the same option mentioned above

### SHEMAT-Suite on vSMP (ScaleMP)

- Geothermal Simulation Package to simulate groundwater flow, heat transport, and the transport of reactive solutes in porous media at high-temperatures (3D)
- Written in Fortran, two levels of parallelism (Nesting):
  - Independent computations of Directional Derivatives (DD)
  - Setup and Solving of Linear Equation Systems (LES)
- Problem with nested OpenMP parallel regions: There is no way to specify the mapping of inner teams to cores, which is quasi-random by default:



- Thread-to-core mapping is performed by a library we created with hierarchical architectures in mind
- Programmer specifies strategies to denote how the threads should be placed throughout the machine

```
#pragma omp parallel for num_threads(4) binding_strategy(scatter)

#pragma omp parallel for num_threads(4) binding_strategy(compact)
//do work
```

- Instrumentation tool adds the library to the application as if it would be part of the OpenMP language
- Evaluation on the ScaleMP machine: 13 boards à 2x Intel Xeon E5420 @ 2.5 GHz connected via 4x DDR InfiniBand  
Single System Image: unmodified Linux kernel (SMP)

