JÜLICH
FORSCHUNGSZENTRUM

Optimisation and Benchmarking

Part 2 – OpenMP

13. Feb 2014|

Alan O‘Cais
a.ocais@fz-juelich.de

Mitglied der Helmholtz-Gemeinschaft

Warning!!
– OpenMP is not as simple as it first appears…

- To write an MPI code you need to work on the whole code, distributing the data and the work completely
- With OpenMP you can develop the code incrementally, but you must work on the whole application in order to get speedup
- OpenMP parallelism is not just about adding a few directives
  - *You don't have to think as deeply as with MPI in order to get your code working*
  - *You* ***do*** *have to think as deeply as with MPI if you want to get your code performing!*
- There are many performance issues that need to be considered

Mitglied der Helmholtz-Gemeinschaft

Hands-on: OpenMP, Pt. 1

- Go to directory `~/PRACE_WinterWorkshop_2014/OpenMP`
- Take a look at `ST1_serial_Frob.c`
- Use `./compile.sh` to compile
- Run it with `./run.sh <arguments>`, check time
- Use `X=40000 Y=40000` as dimensions
- Make a copy, edit original
- Do OpenMP parallelisation with one line

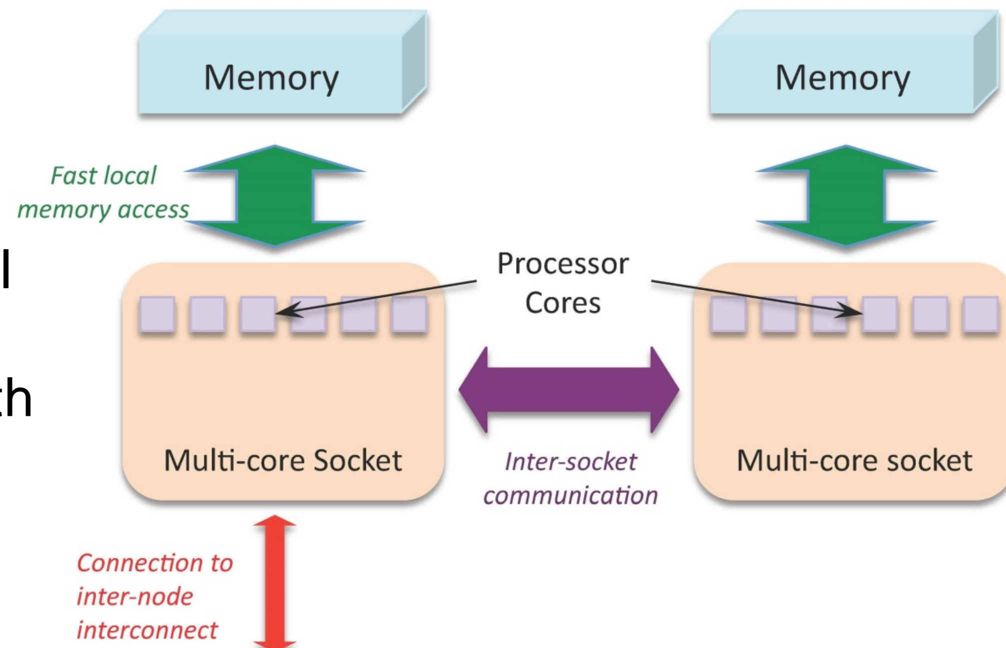
```
#pragma omp parallel for reduction(+:<>) private(<>)
```

- Run it with `./run.sh <arguments>`, check time

NUMA – Non-Uniform Memory Access

Example two-socket node

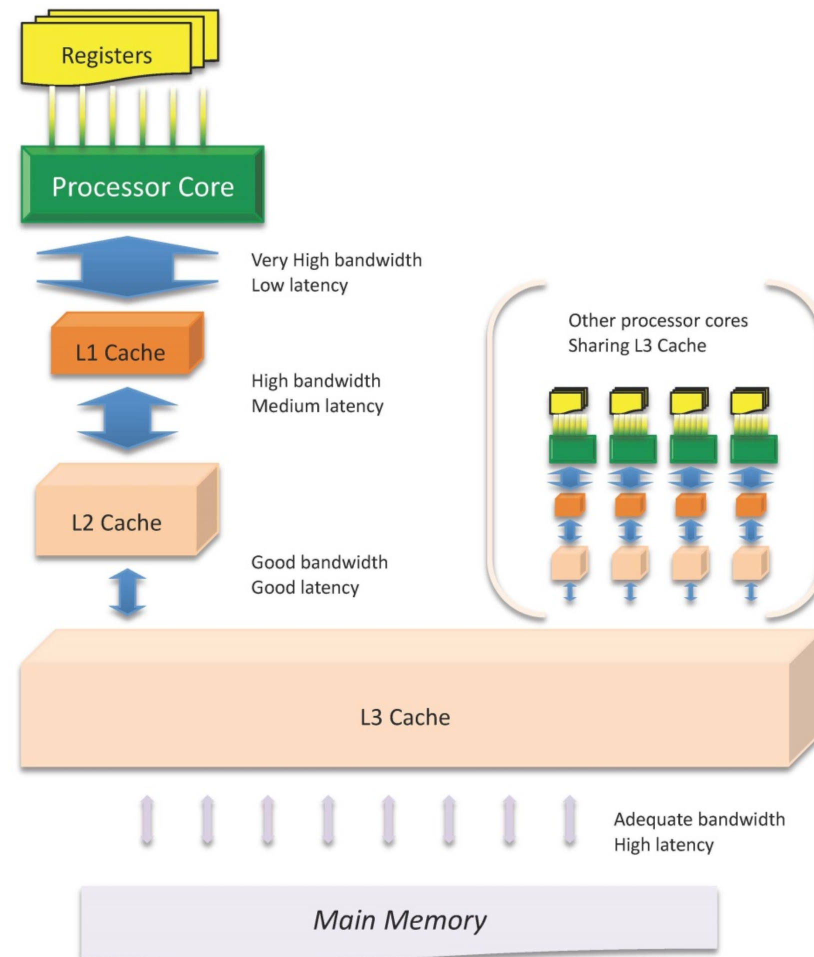
- Local memory accesses have higher bandwidth and lower latency than remote accesses
- If all memory accesses from all cores are to one memory then the effective memory bandwidth is reduced across all processes/threads
- If all accesses are to remote memory then “memory bandwidth” will actually be dominated by inter-socket bandwidth



Mitglied der Helmholtz-Gemeinschaft

Caches Hierarchies and Locality

- Since accessing main memory is slow, modern processors provide fast local memory (cache) to speed up memory accesses
  - *Caches are only effective for data that is being reused*
- Data that has been used recently may have a high likelihood of being used again (temporal locality)
  - *Recently used data sits in the cache in case it is required again soon*
- Data is fetched from main memory to the cache in blocks called **cache lines** as there is a high likelihood that data nearby will be used together (spatial locality)
  - *Often an algorithm will step through adjacent locations in memory*
- There may be multiple levels of cache, each with different characteristics
  - *Most modern processors have 3 levels of cache*
  - *Third level cache (L3 cache) is often shared amongst several processor cores*



Mitglied der Helmholtz-Gemeinschaft

JÜLICH FORSCHUNGSZENTRUM

Cache Coherence

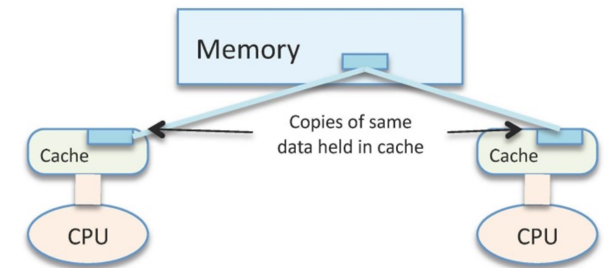
- As caches are local copies of global memory, multiple cores can hold a copy of the same data in their caches
  - *For separate MPI processes with distinct memory address spaces multiple cores are not likely to hold copies of the same user data*
  - *For OpenMP codes where multiple threads share the same address space this could lead to problems*
- Before accessing memory, a processor core will check its own cache and the cache of the other socket to ensure consistency between cache and memory
  - *This is referred to as cache-coherency*
- Ensuring that a node is cache coherent does not mean that problems associated with multiple copies of data are completely removed
  - *Data held in processor registers are not covered by coherence*
  - *This lack of coherence can lead to a race condition*

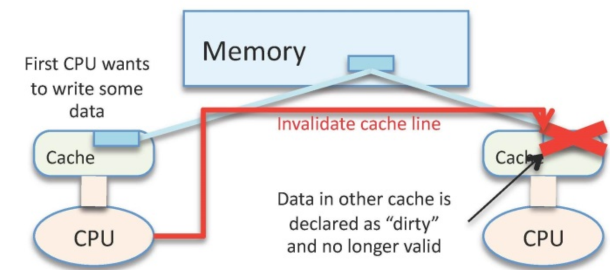
Mitglied der Helmholtz-Gemeinschaft

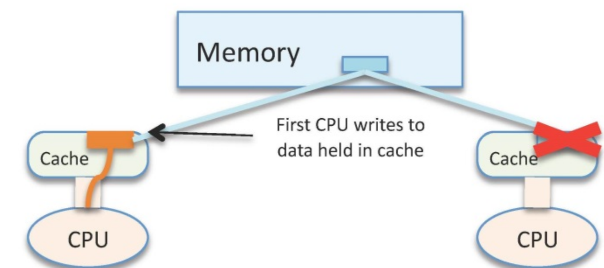
Cache coherence amongst multiple cores

Example two single-core sockets

- Both processors have taken a copy of the same data from main memory
- If one of them wants to write to this data, then the local copy will be affected, but the main memory does not change
  - *The reason for having fast cache is to avoid slower main memory accesses*
- On a cache coherent system, the rest of the node needs to be told about the update
  - Other processor's cache needs to be told that its data is "bad" and that it needs a fresh copy
  - Then the new data can be written into the local copy held in cache
- If CPU 2 wishes to read from or write to the data it needs to get a fresh copy



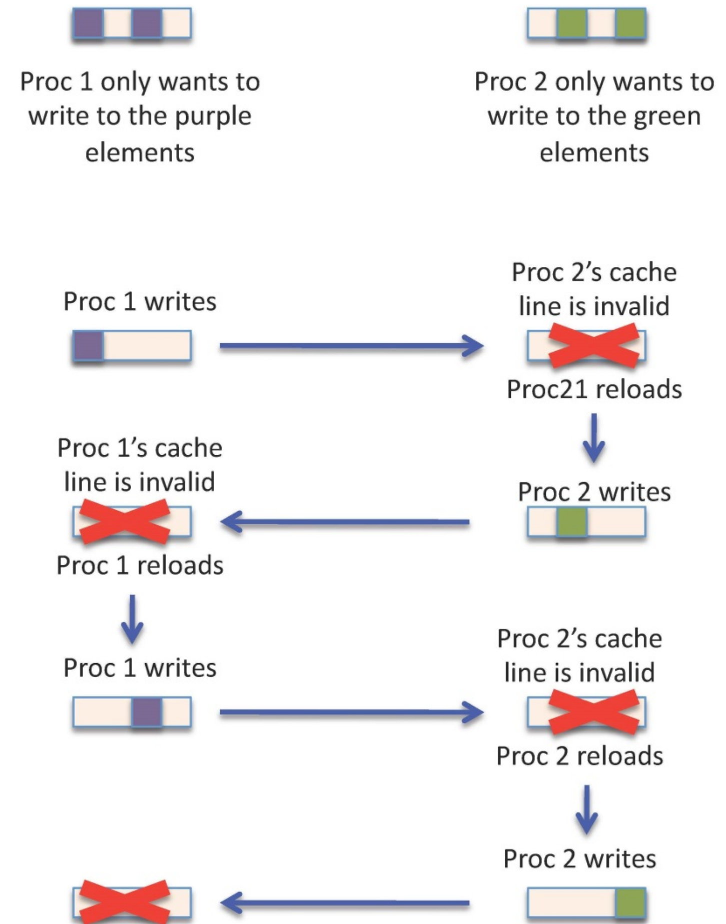




JÜLICH FORSCHUNGSZENTRUM
Mitglied der Helmholtz-Gemeinschaft

Contention - Cache Thrashing, False Sharing

- Cache coherency protocols update data based on cache lines
- Even if two threads want to write to different data elements in an array, if they share the same cache line then cache coherency protocols will mark the other cache line as dirty
- In the best case false sharing leads to serialisation of work
- In the worst case it can lead to slowdown of parallel code compared to the serial version



Mitglied der Helmholtz-Gemeinschaft

Thread creation overhead and synchronisation

- Creation and destruction of threads is an overhead that takes time
- In theory each entry and exit of a parallel region could lead to thread creation and destruction
  - *in most OpenMP implementations threads are not destroyed at the end of a parallel region but are merely put to sleep*
- Entering and exiting a parallel region requires barriers to be called between a team of threads
  - *Often referred to as thread creation/destruction overhead*
- Staying within a parallel region, and having multiple worksharing constructs within it reduces the overhead associated with entering and exiting parallel regions
- For best performance avoid unnecessary synchronisation and consider using NOWAIT with DO/for loops wherever possible

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Hands-on: OpenMP, Pt. 2

- Take a look at `ST3_OMP_Frob_correct.c`
- Use `./compile.sh` to compile
- Run it with `./run.sh <arguments>`, check time
- Now time it with `X=10 Y=40 000 000`
- Compare it with `X=40 000 000 Y=10`
- Why is there a difference?

Mitglied der Helmholtz-Gemeinschaft

Hands-on: OpenMP, Pt. 3

- Now let's get clever
  - Make the code OpenMP independent
  - Allow for reuse of random seed
- Implemented in `ST4_OMP_Frob_persistent_seeds.c`
- Run it with `./run.sh <arguments>`, check time
- What's wrong?
- Fix it using a `private` variable in the parallel region

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Special thanks:
Neil Stringfellow, iVEC Perth (formerly CSCS)

- All slide content from one of his presentations
- See full presentation at:
  - http://www.speedup.ch/workshops/w39_2010/slides/SpeedupTutorial2010Stringfellow.pdf

Mitglied der Helmholtz-Gemeinschaft