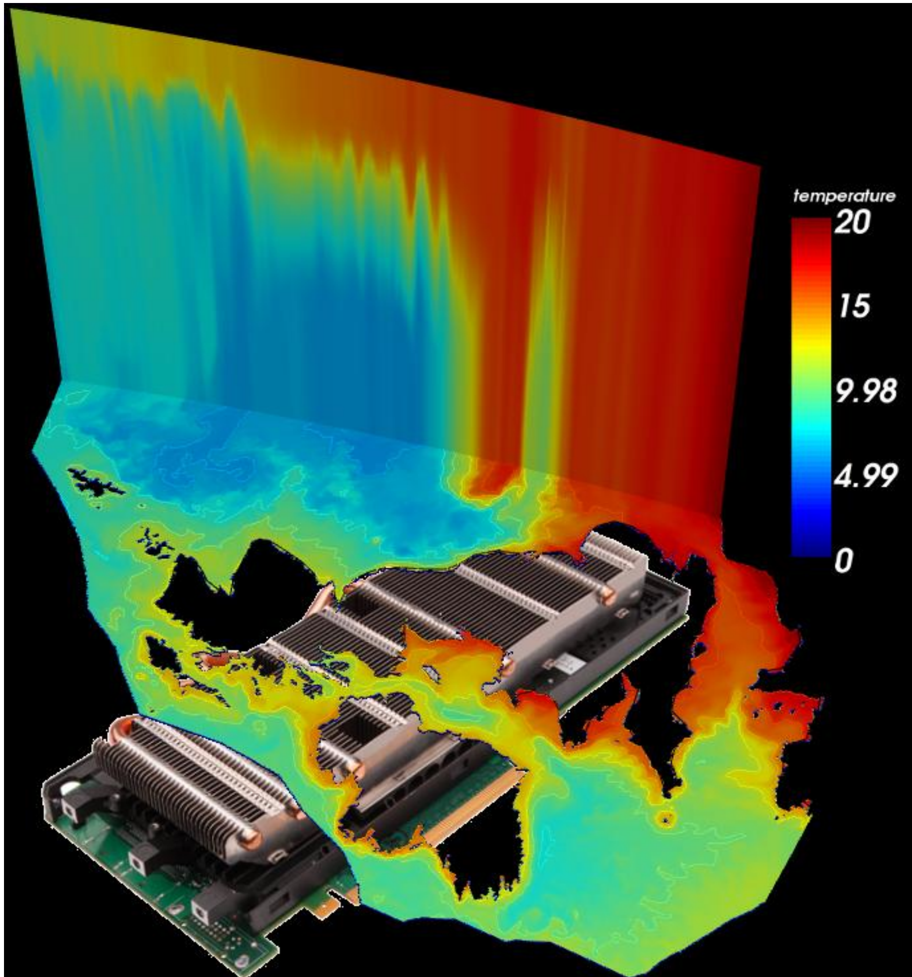
Science & Technology
Facilities Council

Porting a Fortran Oceanographic code to GPUs; the gNEMO Project

Andrew Porter*, Stephen Pickles & Mike Ashworth*

Computational Science & Engineering Dept.
STFC Daresbury Laboratory



Contents

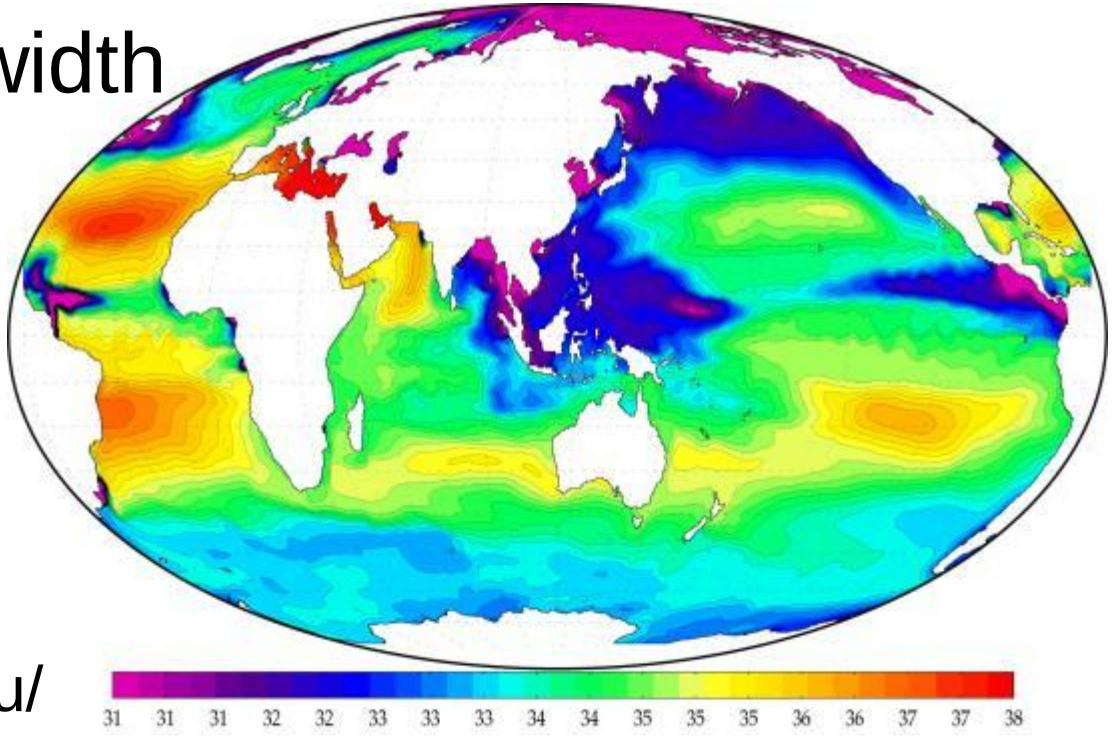
- Oceanography – NEMO
- Accelerator Directives
- Moving a Routine to a GPU
- Performance Results
- Conclusions and Future Prospects
- Acknowledgements

Contents

- Oceanography – NEMO
- Accelerator Directives
- Moving a Routine to a GPU
- Performance Results
- Conclusions and Future Prospects
- Acknowledgements

NEMO

- Widely-used European ocean model
- Fortran90 and MPI
- Highly portable
- Memory-bandwidth bound
- ~20 years of development



http://www.nemo-ocean.eu/

Contents

- Oceanography – NEMO
- Accelerator Directives
- Moving a Routine to a GPU
- Performance Results
- Conclusions and Future Prospects
- Acknowledgements

Accelerator Directives - Motivation

- CUDA & OpenCL are C based
- NEMO core is ~100K lines of Fortran90
- Performance
  - – GPUs have ~10x peak memory bandwidth of a CPU
  - – Maintain single code base but add option to use GPU if available
- Portability
  - – not every computer has a GPU attached

Accelerator ‘Directives’ - Options

Approach	Notes	Fortran support
PGI Accelerator Directives	Currently NVIDIA specific, basis for OpenAcc	Yes
HMPP Workbench	Can generate CUDA and OpenCL code, will support Intel MIC in 2012.	Yes
PGI CUDA Fortran	NVIDIA specific	Yes
OpenCL	Portable, open standard	No
CUDA C	Widely used, mature and low-level but NVIDIA specific	No

Science & Technology Facilities Council

Contents

- Oceanography – NEMO
- Accelerator Directives
- Moving a Routine to a GPU
- Performance Results
- Conclusions and Future Prospects
- Acknowledgements

Porting a Routine I

- Mark-up region to accelerate
  - (Move region into a separate ‘codelet’)
- (In-line any routine calls in region)
- Make all loops explicit
  - no *array(:,:,jk)* notation permitted
- Mark-up the loops to parallelize
- Permute loops for memory coalescing
  - Want consecutive threads to work on consecutive memory addresses

Code Example

```
DO jn = 1, kjpt

 zdit(1,:,:)=0.e0_wp

 DO jk = 1, jpkm1
  DO jj = 1, jpjm1
   DO ji = 1, jpim1
    zdit(ji,jj,jk) = (ptb(ji+1,jj,jk,jn) -
                     ptb(ji,jj,jk,jn) )*umask(ji,jj,jk)
    END DO
   END DO
  END DO
END DO
```

Code Example

```
DO jn = 1, kjpt

 zdit(1,:,:)=0.e0_wp

 DO jk = 1, jpkm1
  DO jj = 1, jpjm1
   DO ji = 1, jpim1
    zdit(ji,jj,jk) = (ptb(ji+1,jj,jk,jn) -
                    ptb(ji,jj,jk,jn) )*umask(ji,jj,jk)
    END DO
   END DO
  END DO
END DO
```

```
!$hmpp parallel
 DO jk = 1, jpkm1
!$hmpp parallel
  DO jj = 1, jpjm1
   zdit(1,jj,jk)=0.e0
  END DO
 END DO
```

Code Example

```
DO jn = 1, kjpt

 zdit(1,:,:)=0.e0_wp

 DO jk = 1, jpkm1
  DO jj = 1, jpjm1
   DO ji = 1, jpim1
    zdit(ji,jj,jk) = (ptb(ji+1,jj,jk,jn) -
                    ptb(ji,jj,jk,jn) )*umask(ji,jj,jk)
    END DO
   END DO
  END DO
END DO
```

```
!$hmppcg permute jj,ji,jk
DO jk = 1, jpkm1
!$hmpp parallel
 DO jj = 1, jpjm1
!$hmpp parallel
  DO ji = 1, jpim1
   zdit(ji,jj,jk,1) = (ptb(ji+1,jj,jk,1) -
                      ptb(ji,jj,jk,1) )*umask(ji,jj,jk)
   zdit(ji,jj,jk,2) = (ptb(ji+1,jj,jk,2) -
                      ptb(ji,jj,jk,2) )*umask(ji,jj,jk)
  END DO
 END DO
END DO
```

Loop over *jn* pushed inside & unrolled

Porting a Routine II

- Analyse data transfers & work to reduce:
  - Keep constant arrays on the device
  - Asynchronous data transfer & kernel execution
  - For halo swaps, transfer halo regions only
  - Overlap transfers of halos to/from GPU with halo packing/unpacking on host
  - *#include* halo pack/unpack code as can’t call subroutines on GPU

Code Example II

```
    END DO
  END DO
END DO
```

kernel2 (doesn’t change zwi)

```
CALL halo_swap( zwi(:,:,:,1))
CALL halo_swap( zwi(:,:,:,2))

DO jk = 1, jpk, 1
  DO jj = 1, jpj, 1
    DO ji = 1, jpi, 1
```

kernel3

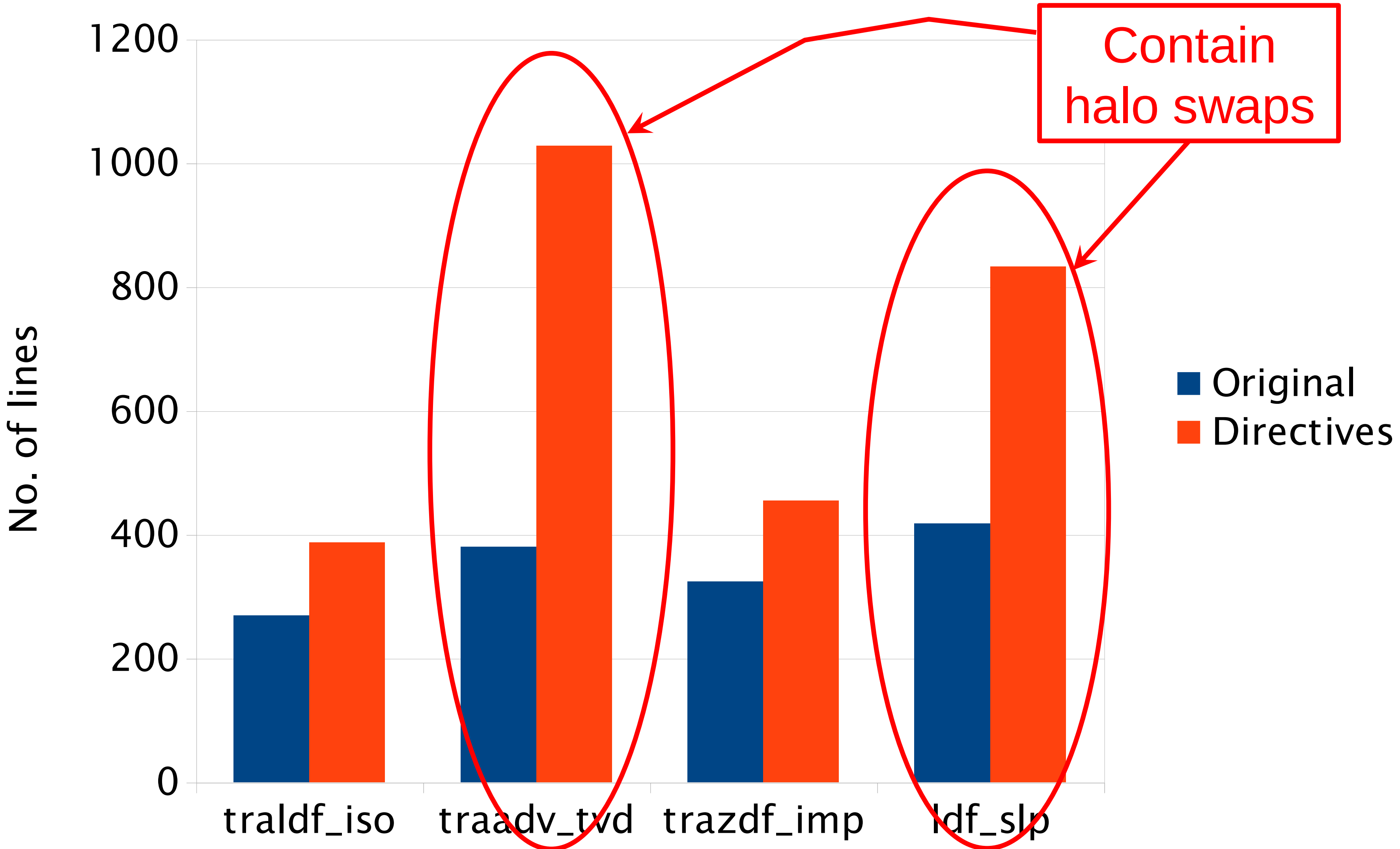
Code Example II

```
!$hmpp <traadv_tvd> kernel1 waitstore,
  args[zhaloswi]
CALL unpack_halos(zhaloswi, zwi, 1)
CALL unpack_halos(zhaloswi, zwi, 2)

CALL halo_swap( zwi(:,:,:,1))
CALL halo_swap( zwi(:,:,:,2))

CALL pack_halos(zhaloswi, zwi, 1)
CALL pack_halos(zhaloswi, zwi, 2)
!$hmpp <traadv_tvd> kernel3 advancedload,
  args[zhaloswi], asynchronous
```

Porting a Routine III



Contain halo swaps

Example – Tracer Advection

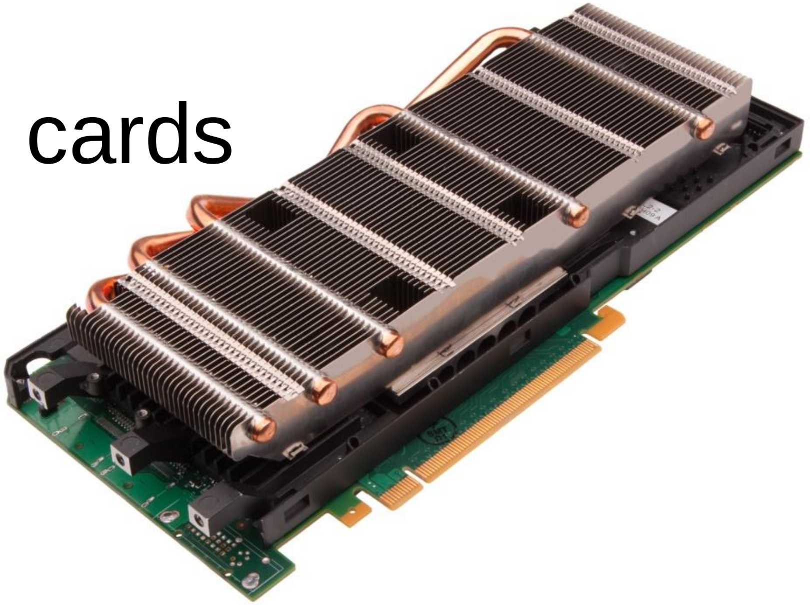
- Originally ~400 lines; GPU version ~1000 lines!
- One child routine (80 lines)
  - Contains one halo swap => splits routine into two codelets
  - Called twice => in-lined twice
- Six separate codelets
  - Six lots of routine interface descriptions
- 16 halo swaps, all for 3D arrays

Contents

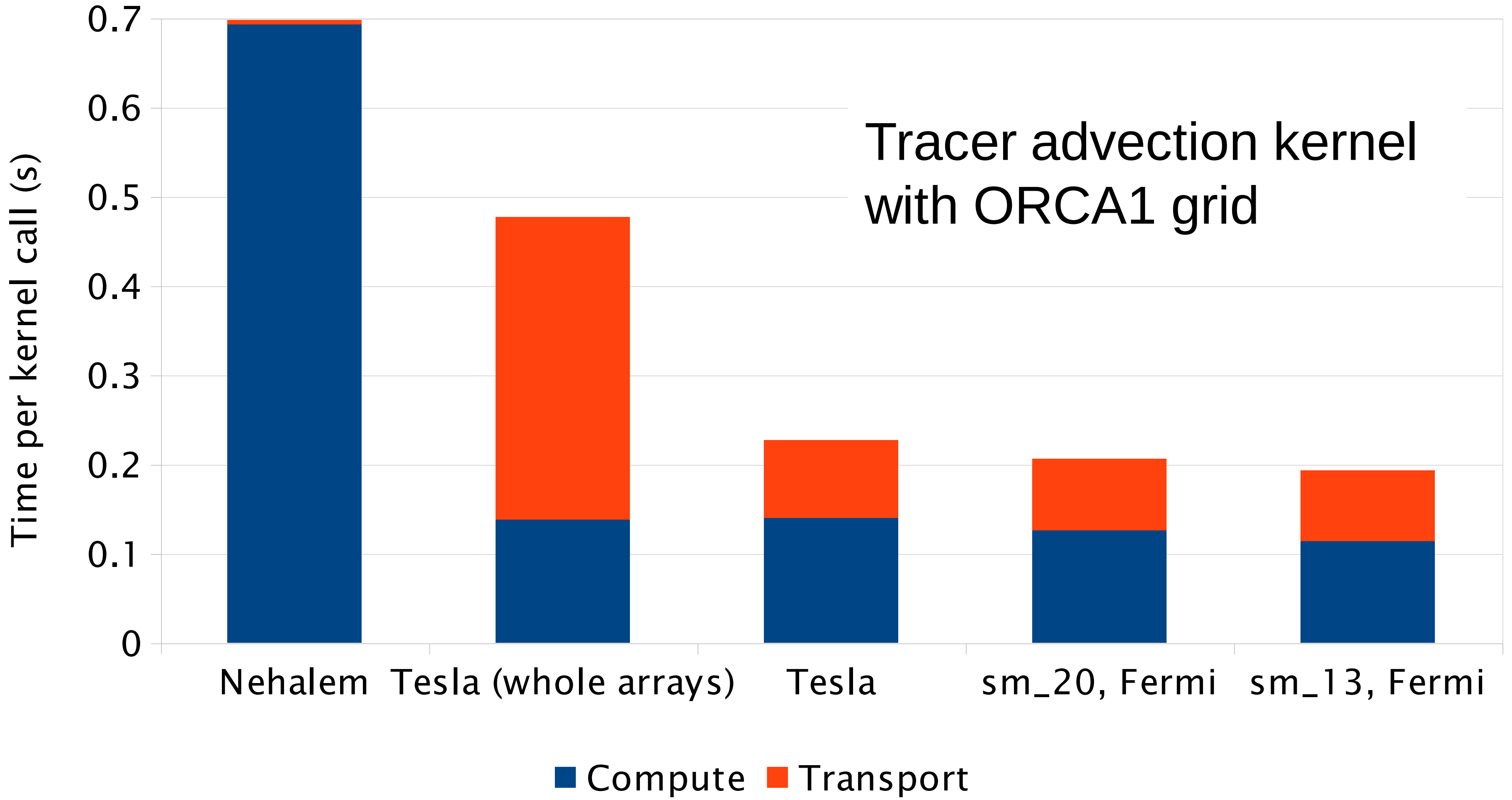
- Oceanography – NEMO
- Accelerator Directives
- Moving a Routine to a GPU
- Performance Results
- Conclusions and Future Prospects
- Acknowledgements

Results - Hardware

- ‘cseht’ & ‘SiD’ machines at Daresbury
- Quad-core Intel Nehalem processor
  - E5540 @ 2.53GHz
- NVIDIA S1070 server
  - contains four M1060 cards
  - ‘Tesla’
- NVIDIA M2050
  - ‘Fermi’

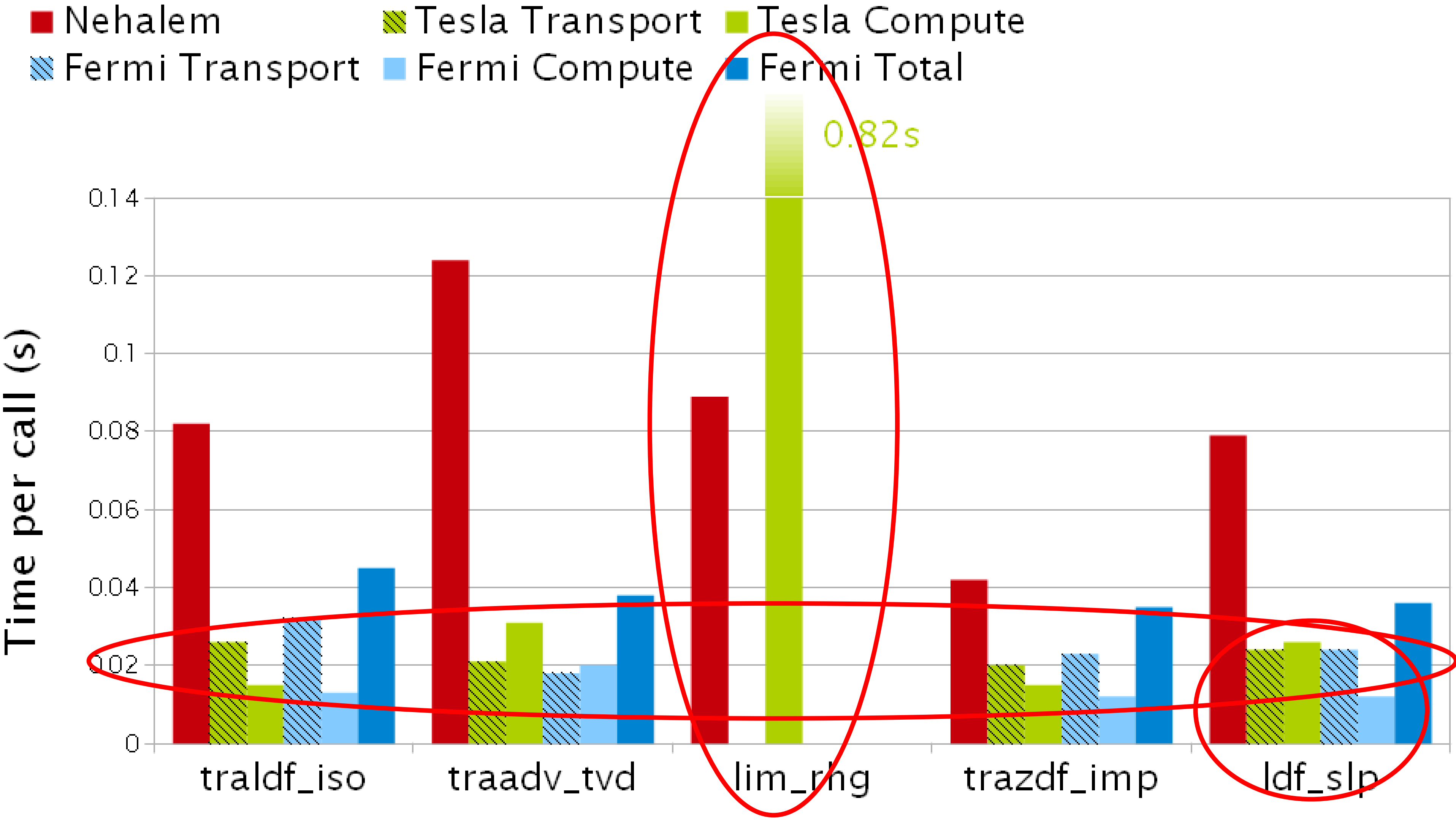


Optimising data transport
- transfer halo regions only



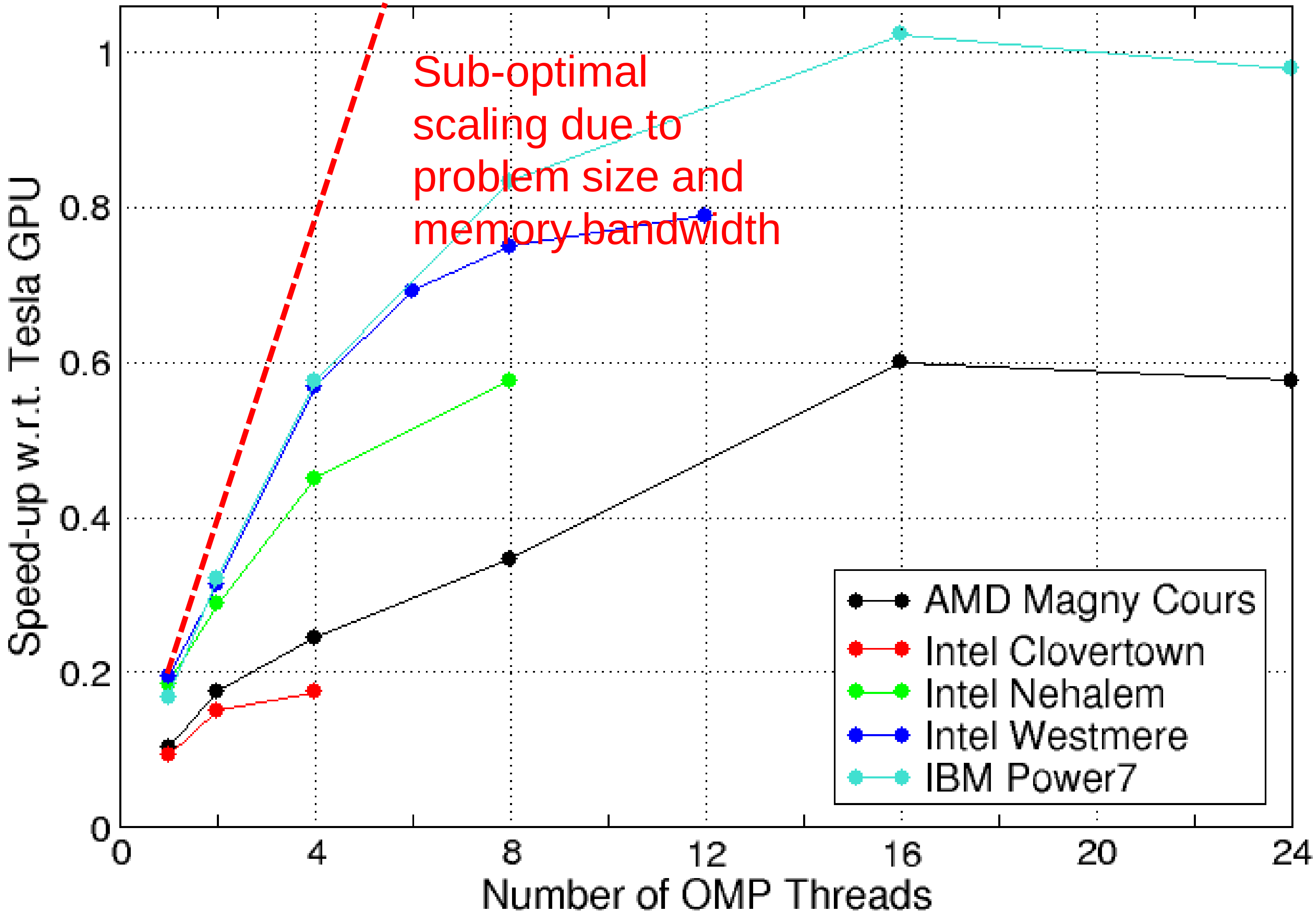
Tracer advection kernel with ORCA1 grid

Kernel Timings



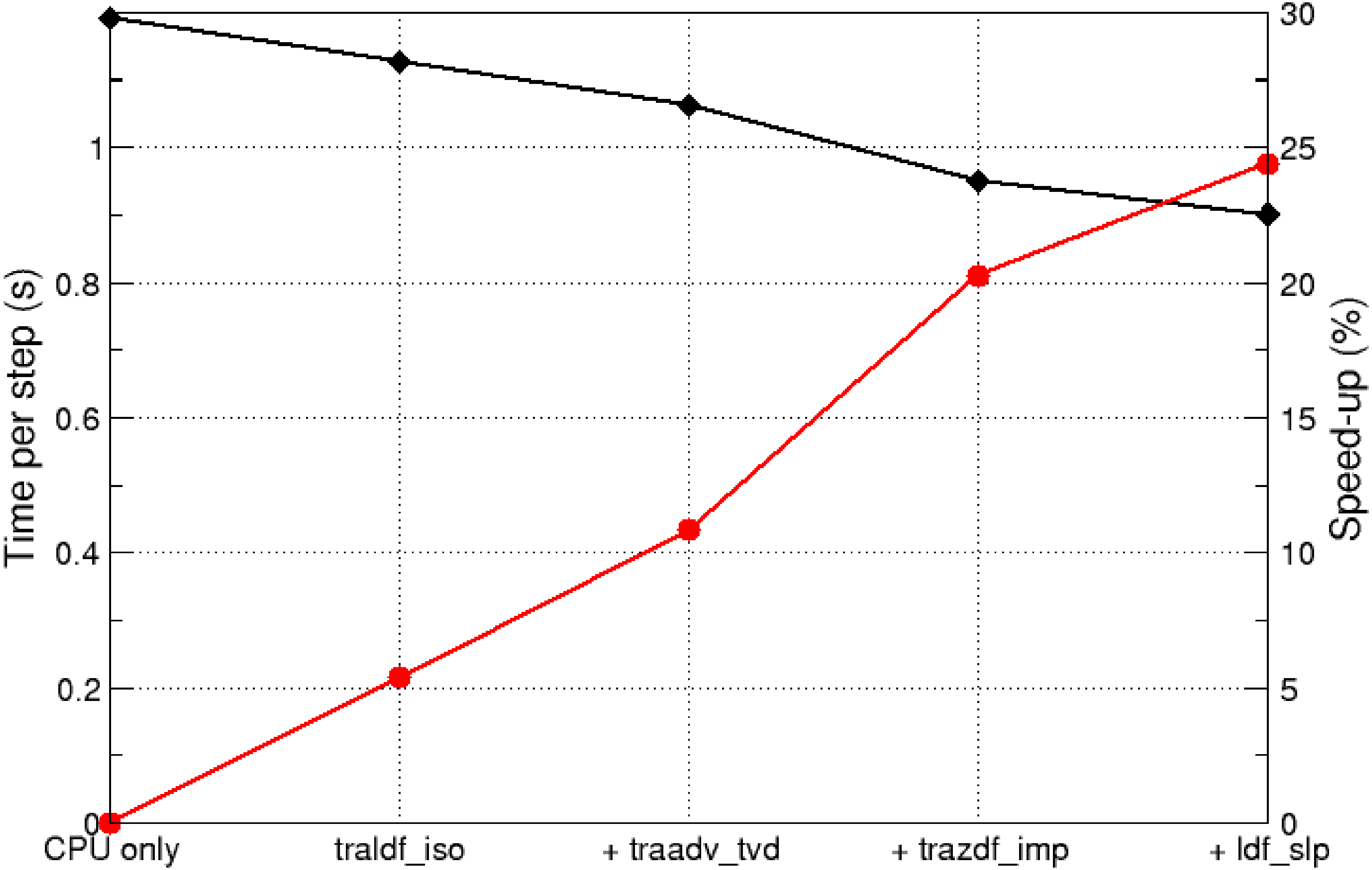
Comparison with OMP

(ORCA2, traldf iso compute only)



Sub-optimal scaling due to problem size and memory bandwidth

Integration with NEMO



Contents

- Oceanography – NEMO
- Accelerator Directives
- Moving a Routine to a GPU
- Performance Results
- Conclusions and Future Prospects
- Acknowledgements

Conclusions

- Successfully ported four routines to GPU using HMPP Workbench
- No speed-up for the sea-ice routine
- Basic porting is fairly straightforward
  - Have to in-line subroutines
  - MPI calls must be on host
  - Can also end up restructuring for performance
- Must work hard to reduce data transfers
- Fragile code

Future Prospects I

- hmpp currently the most mature directives option
  - Also supports asynchronous data transfers
  - Soon to support Intel MIC
- OpenACC announced at SC11
  - Based on PGI’s model
  - PGI, nvidia, Cray & CAPS
  - Initial spec. quite basic

Future Prospects II

- GPUDirect & multi-GPU codes
  - Share pinned memory with Infiniband interconnect
  - DMA between GPUs
  - Avoids doing a copy in system memory for MPI calls
- Move the GPU onto the motherboard
  - Nvidia's Denver, AMD's Fusion, Intel's MIC
  - A single memory address space?

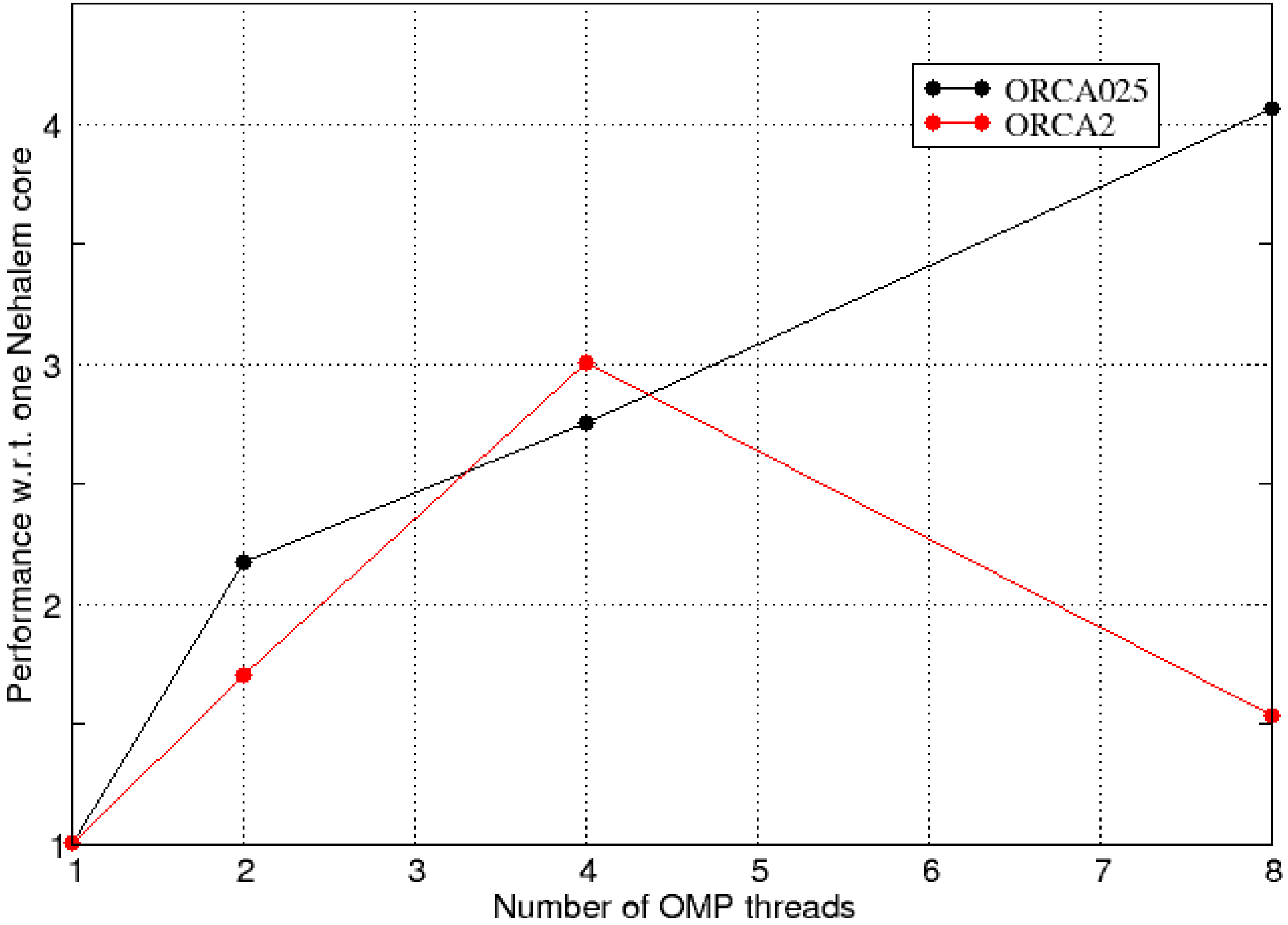
Acknowledgments

- NERC for funding gNEMO
- DiSCO at Daresbury for systems
- Igor Kozin, Xiaohu Guo & Stephen Pickles for advice/discussions
- PGI technical support & forum
- CAPS technical support

NATURAL ENVIRONMENT RESEARCH COUNCIL

Extras...

Scaling of OMP version of lim _rhg


ORCA025
ORCA2
Performance w.r.t. one Nehalem core
Number of OMP threads