OpenMP Users' Conference 2018

The TORUS radiative transfer code: OpenMP implementation and performance

David Acreman, University of Exeter
Tim Harries, University of Exeter
Tom Haworth, Imperial College London

Introduction: TORUS

- Transport Of Radiation Using Stokes intensities
- An astrophysical radiation-hydrodynamics code
- Core algorithm is Monte-Carlo Radiation Transport on an adaptive grid
- Written in Fortran (modern style) and parallelised with OpenMP and MPI

Overview

1. Background and astrophysical applications

2. Algorithms and parallelisation

3. Parallel performance

Star formation

- Stars form from collapsing clouds of gas and dust (turbulence and self-gravity)

- Young stars are surrounded by a disc of gas and dust from which planets form

- Massive stars generate winds and radiation which influences their environment

- TORUS has been applied to a wide range of star formation problems

Credits: NASA, ESA and the Hubble Heritage Team (STScI/AURA)
https://www.nasa.gov/feature/goddard/2017/messier-16-the-eagle-nebula

Challenges in star formation simulations

- Need to include a wide range of physics:
  - hydrodynamics
  - radiative feedback
  - self-gravity
  - magnetic fields
- Wide range of densities and spatial scales
- Complex geometries

Radiation transport

- When stars form they start to emit radiation (gravitational collapse then nuclear burning)

- Radiation heats surrounding material and influences its subsequent evolution

- Ionising radiation produces plasma which interacts with magnetic fields

- Also need to understand radiation transport to interpret observations: visible light, radio, infra-red, x-rays etc.

- Cases of practical interest are too complex to be solved analytically and are treated numerically

Spatial representation: adaptive grid

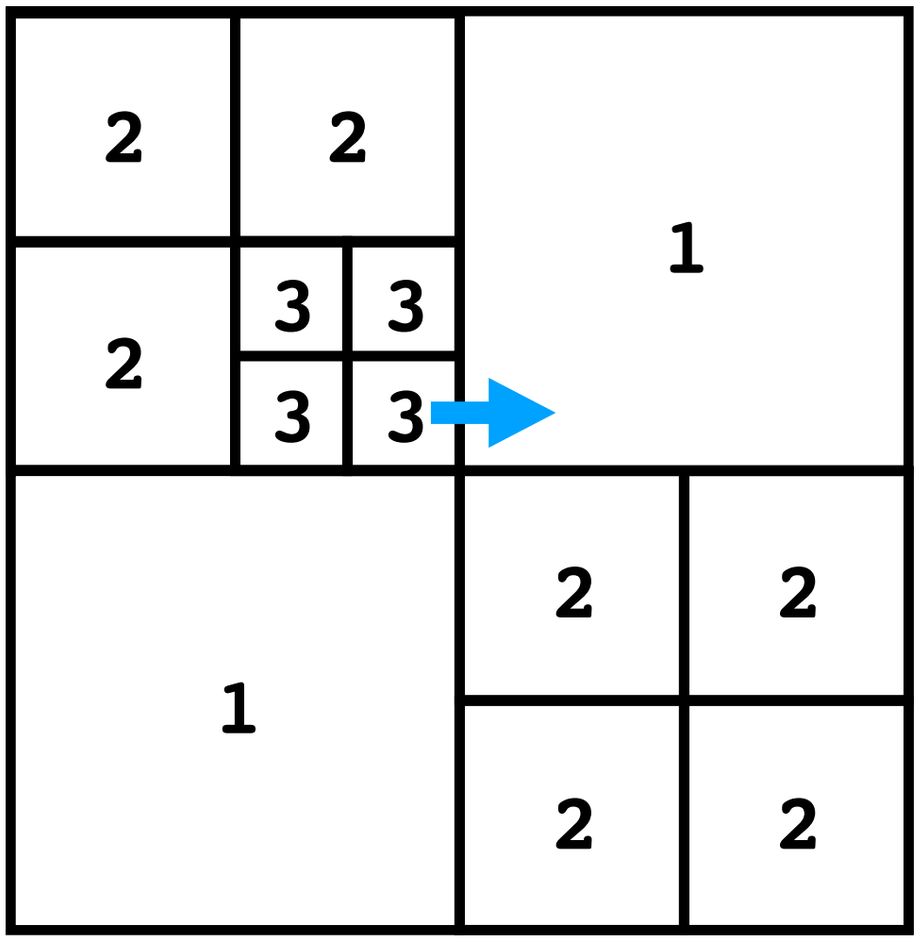
- Star formation processes occur on a large range of spatial scales

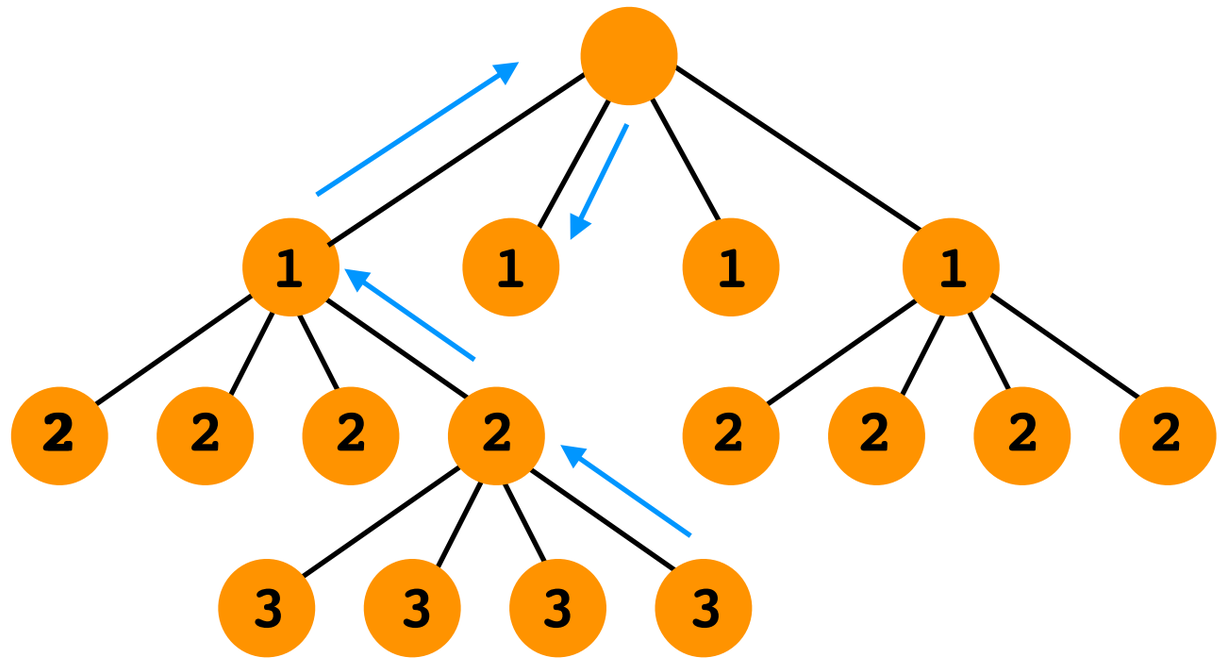
- The spatial scale which determines key properties of radiative transfer can be very small compared to entire computational domain

- Radiative transfer needs to resolve edges (opacity gradients)

- Use an adaptive spatial grid to give sharp edges and wide range of spatial resolution

- Tree structured AMR: flexible representation - refine individual cells as required





Cells (voxels) in grid are leaf nodes of the tree

Grid operations

- “Octal” derived type holds physical variables and pointers to child octals
- Traverse tree structure with recursive subroutine calls to follow pointers
- Carry out operations on leaf nodes

```
recursive subroutine zeroAdot(thisOctal)
type(octal), pointer   :: thisOctal
type(octal), pointer  :: child
integer :: subcell, i

do subcell = 1, thisOctal%maxChildren
     if (thisOctal%hasChild(subcell)) then
        do i = 1, thisOctal%nChildren
        if (thisOctal%indexChild(i) == subcell) then
           child => thisOctal%child(i)
           call zeroAdot(child)
           exit
        end if
        end do
     else
        thisOctal%aDot(subcell) = 0.d0
     endif
  enddo
end subroutine zeroAdot
```

Monte-Carlo Radiative Transfer

- Core algorithm in TORUS is Monte-Carlo Radiative Transfer*

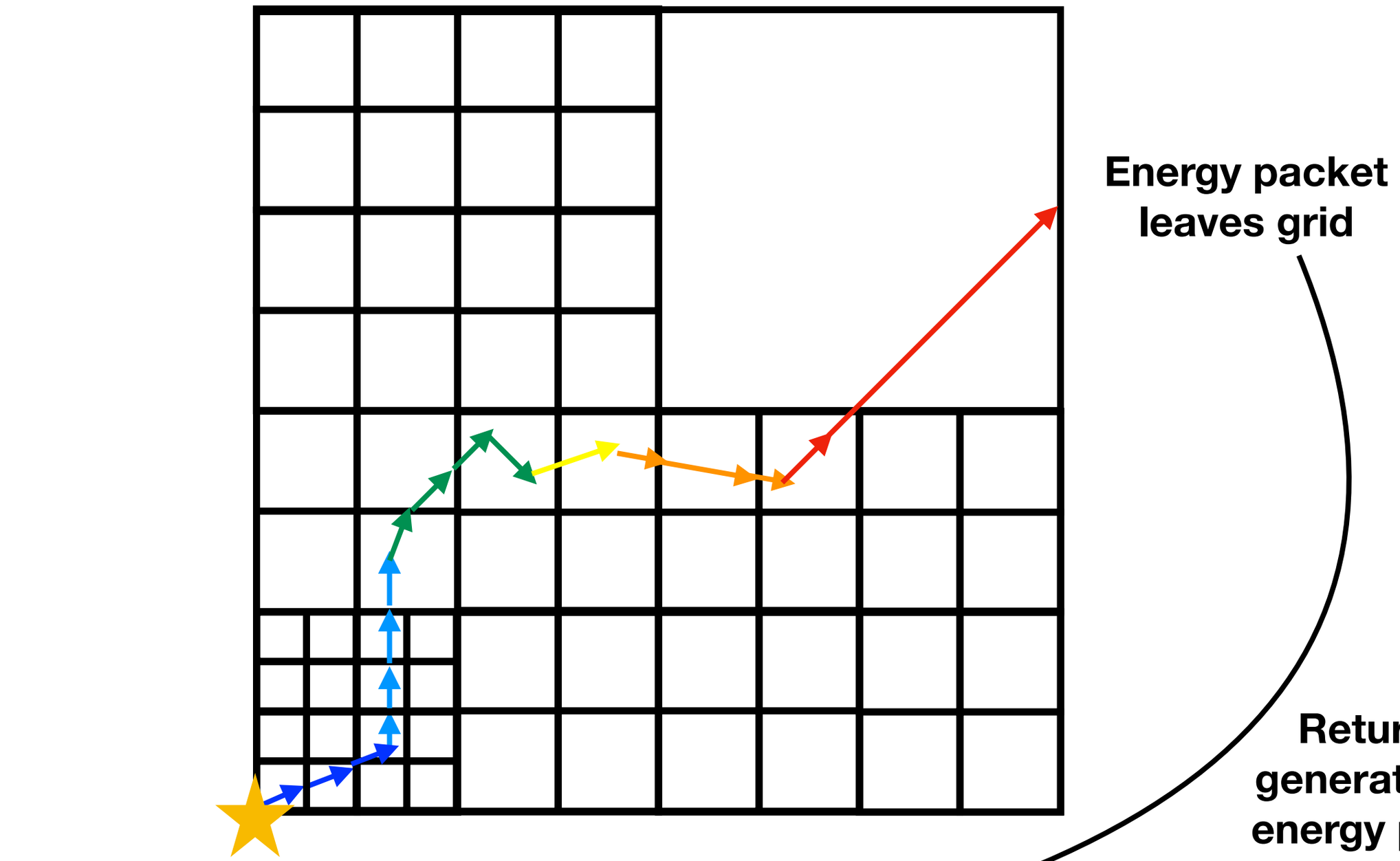
- A energy packets are propagated through the computational domain in random directions from sources

- Energy packets interact with dust and gas:

  - scattering: change direction

  - absorption: change direction and wavelength (depends on local temperature)

*** Lucy L. B., 1999, Astronomy & Astrophysics, 344, 282**


Energy packet
leaves grid
Return to
generate next
energy packet
Energy packet
generated by source
with random direction
and wavelength

Monte-Carlo Radiative Transfer

- Energy density (hence temperature) is calculated from the time an energy packet spend in a given grid cell

- Iterate to converge temperatures and radiation fields (re-emission after absorption depends on temperature)

+ Enables treatment of complex geometries

+ Cells are sampled when they are crossed → sample optically thin regions

– Can be computationally expensive to achieve well converged solution

OpenMP Parallelisation

- Fortunately the algorithm is amenable to parallelisation
- Each energy packet can be independently propagated through the grid
- `DO` loop over energy packets is parallelised with OpenMP
  - Grid is `SHARED` and updates to octal components are `ATOMIC`
  - Use `DYNAMIC` scheduling ensure load balancing

```
!$OMP ATOMIC
    thisOctal%distanceGrid(subcell) = thisOctal%distanceGrid(subcell) + tVal_db * kappaAbsdb * packetWeight
    if (usePAH) then
!$OMP ATOMIC
       thisOctal%aDotPAH(subcell) = thisOctal%adotPAH(subcell) + tVal_db * getkappaAbsPAH(thisFreq) * packetWeight
    endif
!$OMP ATOMIC
    thisOctal%nCrossings(subcell) = thisOctal%nCrossings(subcell) + 1
```

Performance and scaling study

- Investigate performance without making code modifications → practical advice on how to run TORUS
- Run on Exeter University cluster:
  - 20-cores/node (2x10 core Intel Broadwell @ 2.40GHz)
  - 128GB RAM/node
  - 4X EDR infiniband

Test case: 3D disc benchmark

- Calculates radiative equilibrium in disc of material around a star
- 3D cylindrical polar grid with 320,048 voxels (leaf-nodes)
- 3,200,480 energy packets per iteration
- Representative of a typical TORUS Monte-Carlo calculation and large enough problem to be realistic
- Record average time for an iteration of the MCRT algorithm (>99.9% of run time in serial run without I/O)

Compilers and affinity

- OpenMP performance tests:
  - Intel (ifort 16.0.3) and GNU (gfortran 5.4.0)
  - Compiler flags: optimisation and architecture targeting
  - Affinity: pinning threads to CPU cores using `KMP_AFFINITY` environment variable (Intel compiler)

OpenMP-only: compilers and flags

Normalised to fastest run

Compiler	Flags	Threads	Affinity	Time (s)	Norm. time
gfortran 5.4.0	-O2	20	none	193	1.11
gfortran 5.4.0	-O3	20	none	192	1.11
ifort 16.0.3	-O2	20	none	196	1.13
ifort 16.0.3	-O3	20	none	180	1.04
ifort 16.0.3	-O3 -xCORE-AVX2	20	none	196	1.13
ifort 16.0.3	-O3	20	compact	173	1
ifort 16.0.3	-O3	20	scatter	176	1.01
ifort 16.0.3	-O3	10	compact	216	1.25
ifort 16.0.3	-O3	10	scatter	231	1.36

Compiler optimisations ~10 %

Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz (Broadwell)

OpenMP-only: affinity

Compiler	Flags	Threads	Affinity	Time (s)	Norm. time
gfortran 5.4.0	-O2	20	none	193	1.11
gfortran 5.4.0	-O3	20	none	192	1.11
ifort 16.0.3	-O2	20	none	196	1.13
ifort 16.0.3	-O3	20	none	180	1.04
ifort 16.0.3	-O3 -xCORE-AVX2	20	none	196	1.13
ifort 16.0.3	-O3	20	compact	173	1
ifort 16.0.3	-O3	20	scatter	176	1.01
ifort 16.0.3	-O3	10	compact	216	1.25
ifort 16.0.3	-O3	10	scatter	231	1.36

Affinity ~few %

Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz (Broadwell)

Thread placement on half-populated node

Compiler	Flags	Threads	Affinity	Time (s)	Norm. time
gfortran 5.4.0	-O2	20	none	193	1.11
gfortran 5.4.0	-O3	20	none	192	1.11
ifort 16.0.3	-O2	20	none	196	1.13
ifort 16.0.3	-O3	20	none	180	1.04
ifort 16.0.3	-O3 -xCORE-AVX2	20	none	196	1.13
ifort 16.0.3	-O3	20	compact	173	1
ifort 16.0.3	-O3	20	scatter	176	1.01
ifort 16.0.3	-O3	10	compact	216	1.25
ifort 16.0.3	-O3	10	scatter	231	1.36

NUMA ~10 %

Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz (Broadwell)

DIMMS

CPU

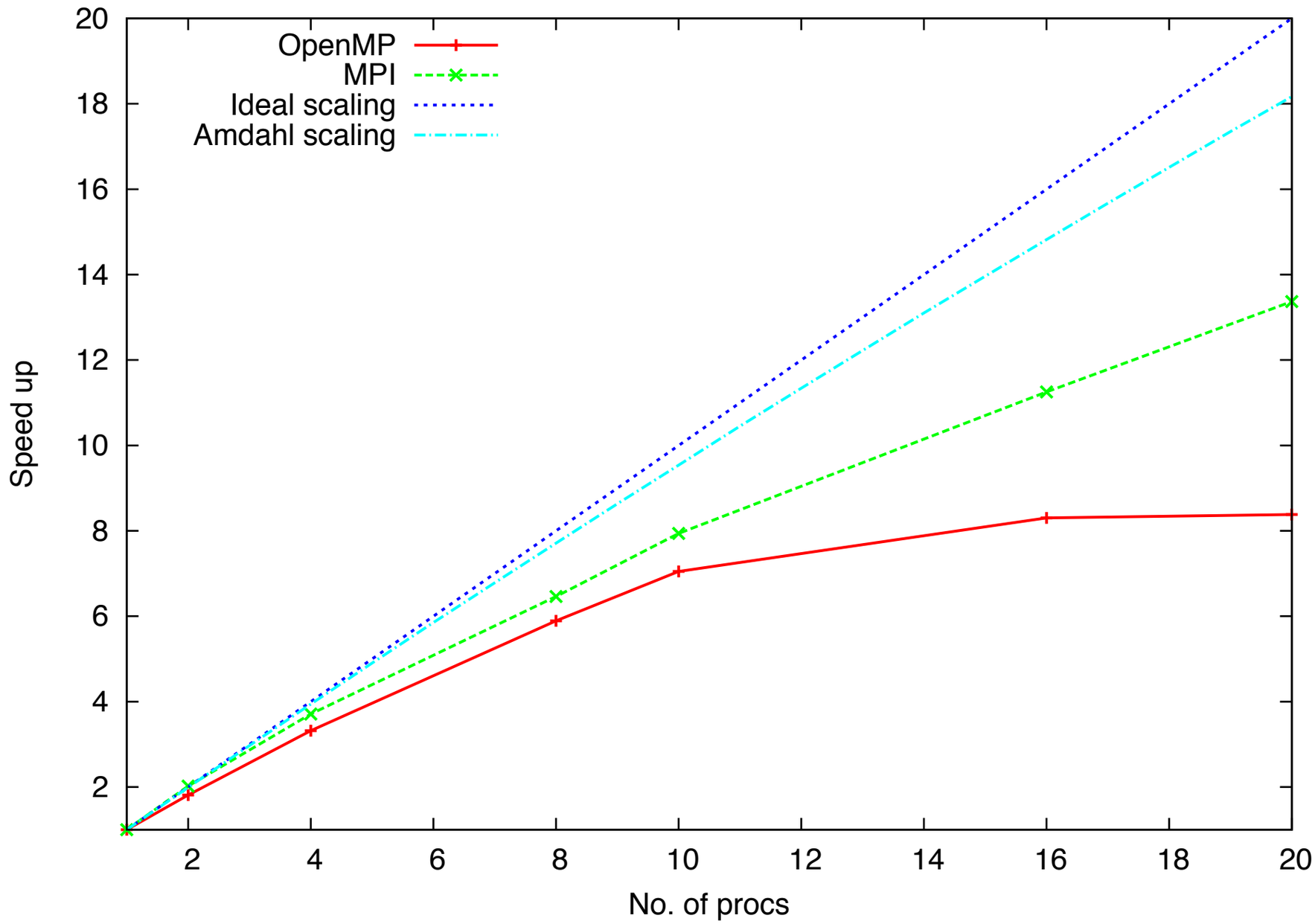
CPU

DIMMS

Compilers and affinity

- Intel compiler with `-O3` and compact affinity gives best single node OpenMP performance
- Performance is better without `-xCORE-AVX2` flag
- Compact affinity better than scatter on half populated node
- Compiler, optimisation and affinity effects ~10% on a fully populated node - potentially a quick performance gain

Single node scaling

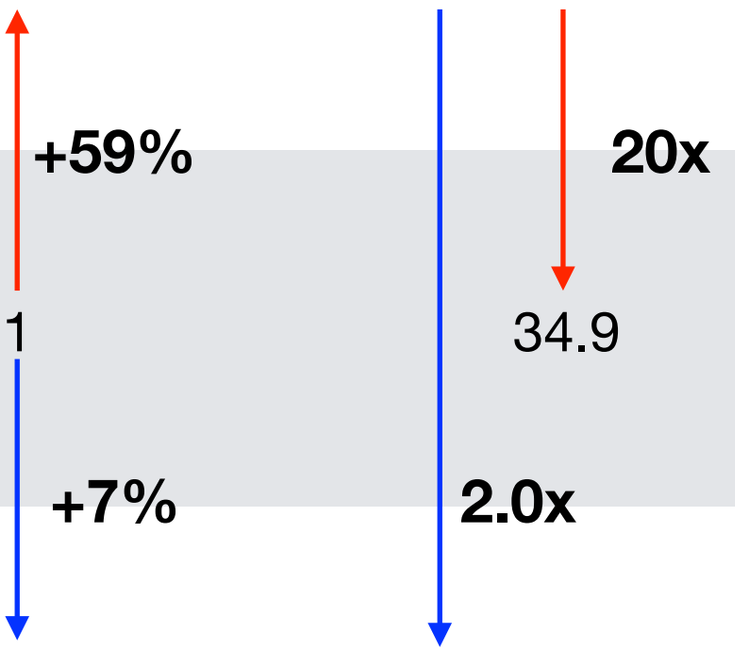


Parallel fraction = 0.99467

Max speed up = 188

Single node OpenMP vs MPI vs hybrid

Parallelism	Time(s)	Normalised time	Memory footprint (GB)
OpenMP (20 threads)	179.9	1.59	1.73
MPI (20 processes)	112.8	1	34.9
Hybrid (2 MPI x 10 OpenMP)	120.8	1.07	3.47

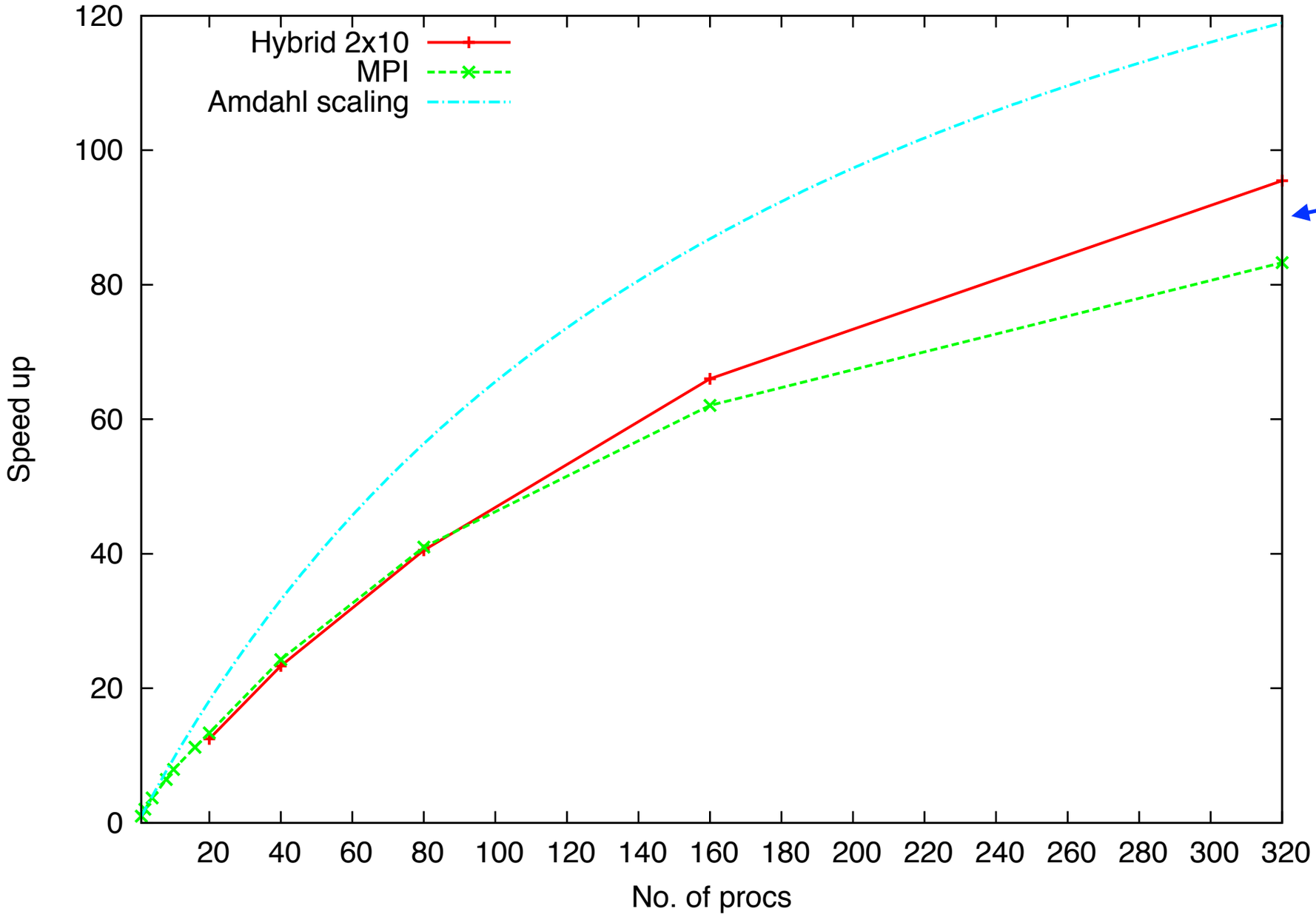


`-genv I_MPI_PIN_DOMAIN socket` (`-genv I_MPI_PIN_DOMAIN omp` for 4 MPI procs per node)

Single node results

- No clear winner for best parallelisation:
  - OpenMP for ease of use and minimum memory foot print
  - MPI gives better performance than OpenMP but memory footprint is much larger
  - Hybrid OpenMP-MPI reduces memory footprint with good performance but harder to set up correctly than OpenMP or MPI alone
- Can’t predict where energy packets will go so can’t partition grid using first touch memory allocation

Multi-node scaling



Hybrid scales better than pure MPI

Parallel fraction = 0.99467

Max speed up = 188

Isambard

- EPSRC tier-2 facility
- GW4 consortium (Bath, Bristol, Cardiff, Exeter Universities), Met Office and Cray
- ARMv8 + KNLs and GPUs

Isambard

- Investigated single node performance on ARMv8
  - 2x 32-core Cavium ThunderX2 @ 2.2 GHz
  - 64 cores x 4 way SMT → 256 h/w threads per node
  - gfortran 6.1.0
  - OpenMP 4.0 affinity mechanism:
    - `export OMP_PLACES=core|thread`
    - `export OMP_BIND=close|spread`

Pure OpenMP on ThunderX2 (ARMv8): affinity

No. of OpenMP	h/w threads per core	PLACES	BIND	Time (s)	Norm. time
256	4	-	-	102	1
256	4	threads	close	111	1.09
256	4	cores	close	116	1.14
256	4	sockets	close	104	1.02
128	2	-	-	103	1.01
128	2	sockets	spread	102	1
128	2	cores	close	102	1
128	2	threads	spread	135	1.32

Pure OpenMP on ThunderX2 (ARMv8)

No. of OpenMP threads	h/w threads per core	PLACES	BIND	Time (s)	Norm. time
256	4	cores	close	116	1.14
128	2	cores	close	102	1
64	1	cores	close	125	1.23
32	1	cores	close	203	1.99
32	1	cores	spread	223	2.19

NUMA ~10 %

Hybrid MPI-OpenMP

- On x86 hybrid performed better than pure OpenMP on a fully populated node. What about ARM?
- OpenMPI 3.0.0, gfortran 6.1
- `mpirun --bind-to socket --npersocket 1 -np 2 ./torus.gfortran`
- `export OMP_PLACES=core; export OMP_BIND=close`

No. of MPI processes	Threads/core	No. of OpenMP threads	Time (s)
2	1	32	113
2	2	64	88
2	4	128	72
4	1	16	169
4	2	32	104
4	4	64	87

Best pure OpenMP time on ARMv8 is 102s

Conclusions

- OpenMP has been very important in TORUS (memory footprint)
- NUMA effects are significant in two socket nodes
- Hybrid OpenMP-MPI (OpenMP within a socket) gives better performance than whole-node OpenMP
- Hybrid OpenMP-MPI scales better than pure-MPI with smaller memory footprint
- Code paper: Harries, Haworth, Acreman and Ali (in prep)