THREADS AND THEIR USE WITH OPENMP

Scaling your science on MIRA
May 25, 2016

ANSHU DUBEY
MATHEMATICS AND COMPUTER SCIENCE DIVISION

PROCESSES & THREADS

- Both are abstractions for control flow during execution
  - They have their context; meta information necessary for running
  - Multiple threads and processes can run on the same hardware through context switching
- They differ in what they share
  - Each process has its own code and data
    - Suitable for distributed memory (MPI) view
  - Threads share code and data
    - Suitable for shared memory view
- Threads are also less expensive

USING THREADS

- A process can have multiple threads
  - Threads associated with a process form a pool of peers

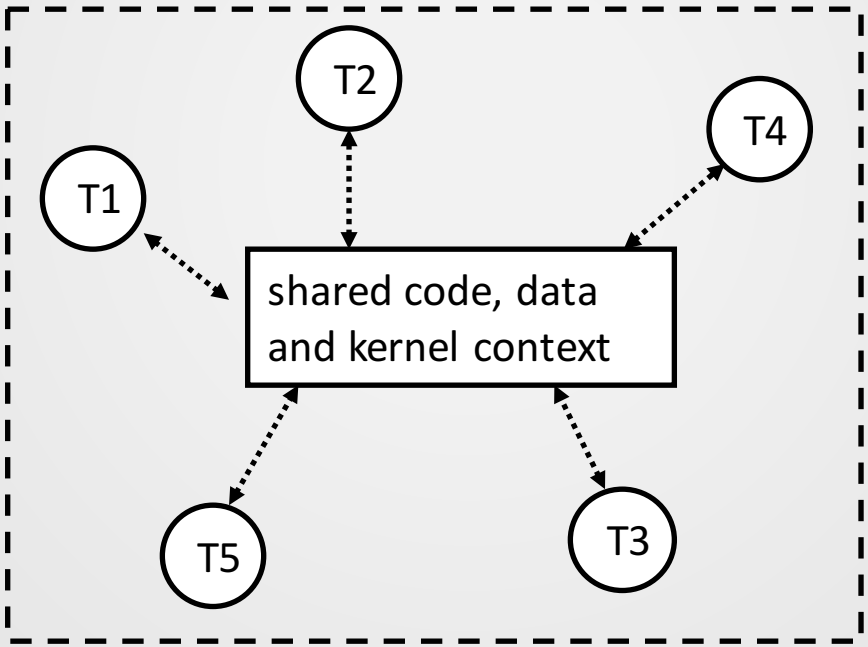
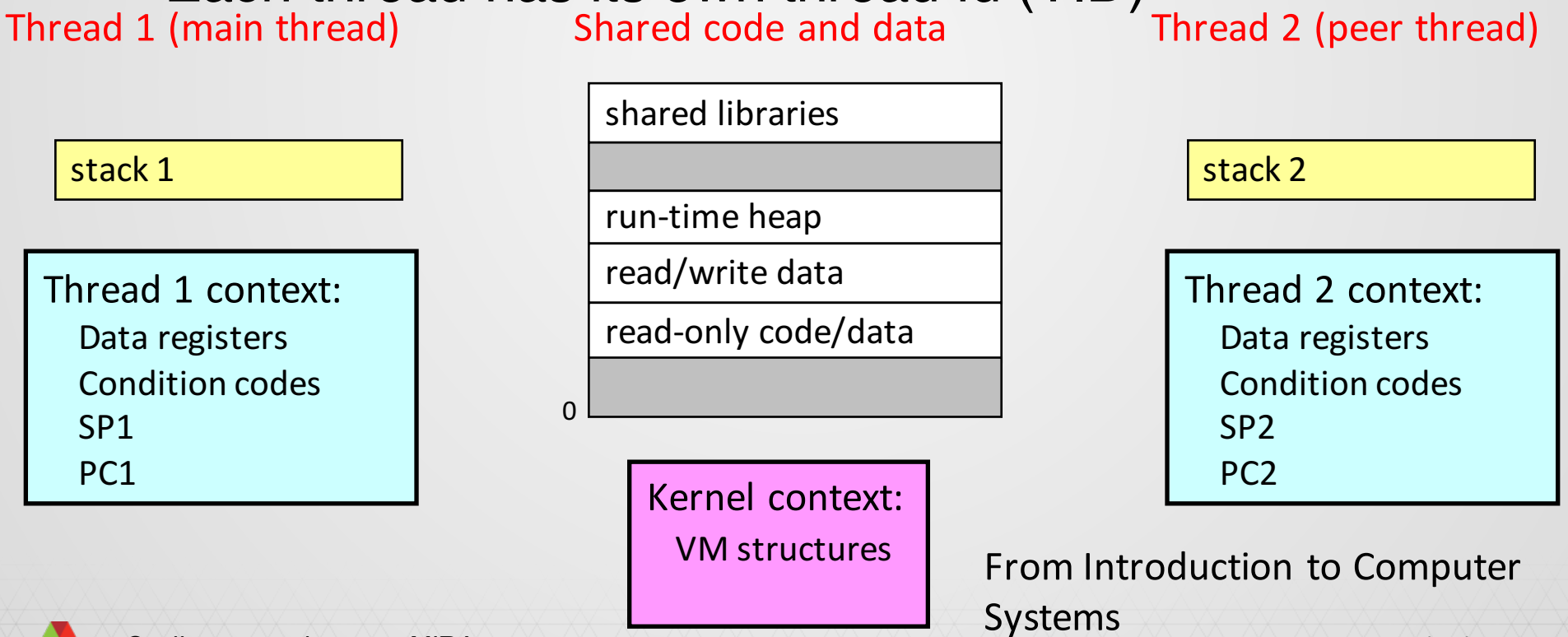


Figure from Introduction to Computer Systems

A VIEW OF SHARED THREADS

- Each thread has its own logical control flow (sequence of PC values)
- Each thread shares the same code, data, and kernel context
- Each thread has its own thread id (TID)



From Introduction to Computer Systems

THREADS MEMORY MODEL

- Conceptual model:
  - Each thread runs in the context of a process.
  - Each thread has its own separate thread context.
    - Thread ID, stack, stack pointer, program counter, condition codes, and general purpose registers.
  - All threads share the remaining process context.
    - Code, data, heap, and shared library segments of the process virtual address space.
    - Open files and installed handlers
- Operationally, this model is not strictly enforced:
  - While register values are truly separate and protected....
  - Any thread can read and write the stack of any other thread.

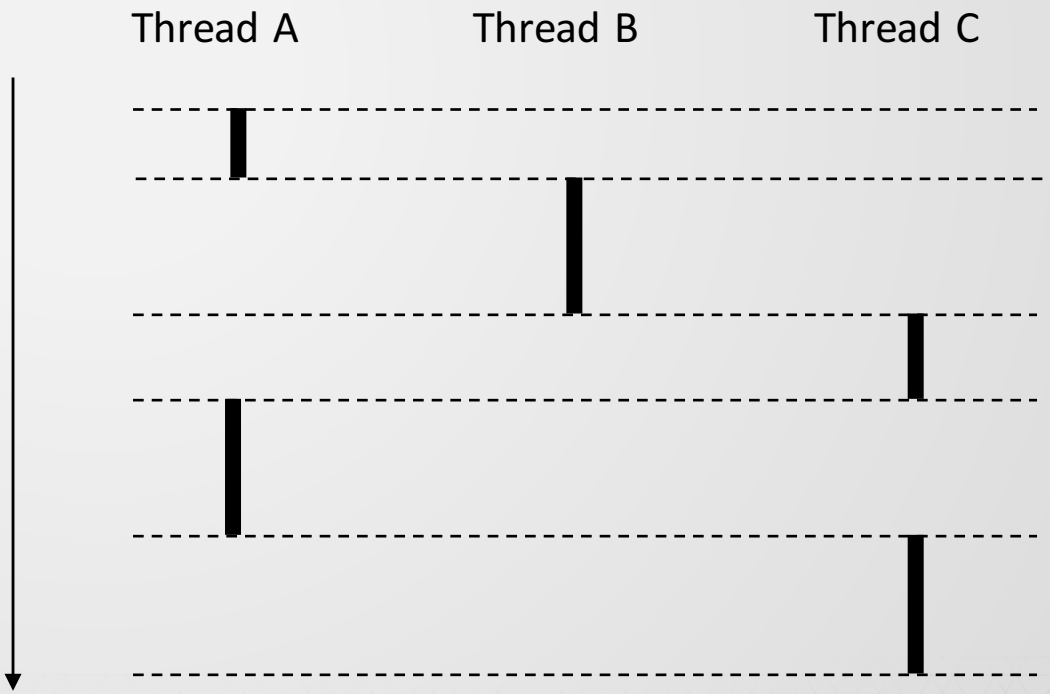
- *Mismatch between the conceptual and operation model is a source of confusion and errors.*

From Introduction to Computer Systems

CONCURRENT EXECUTION

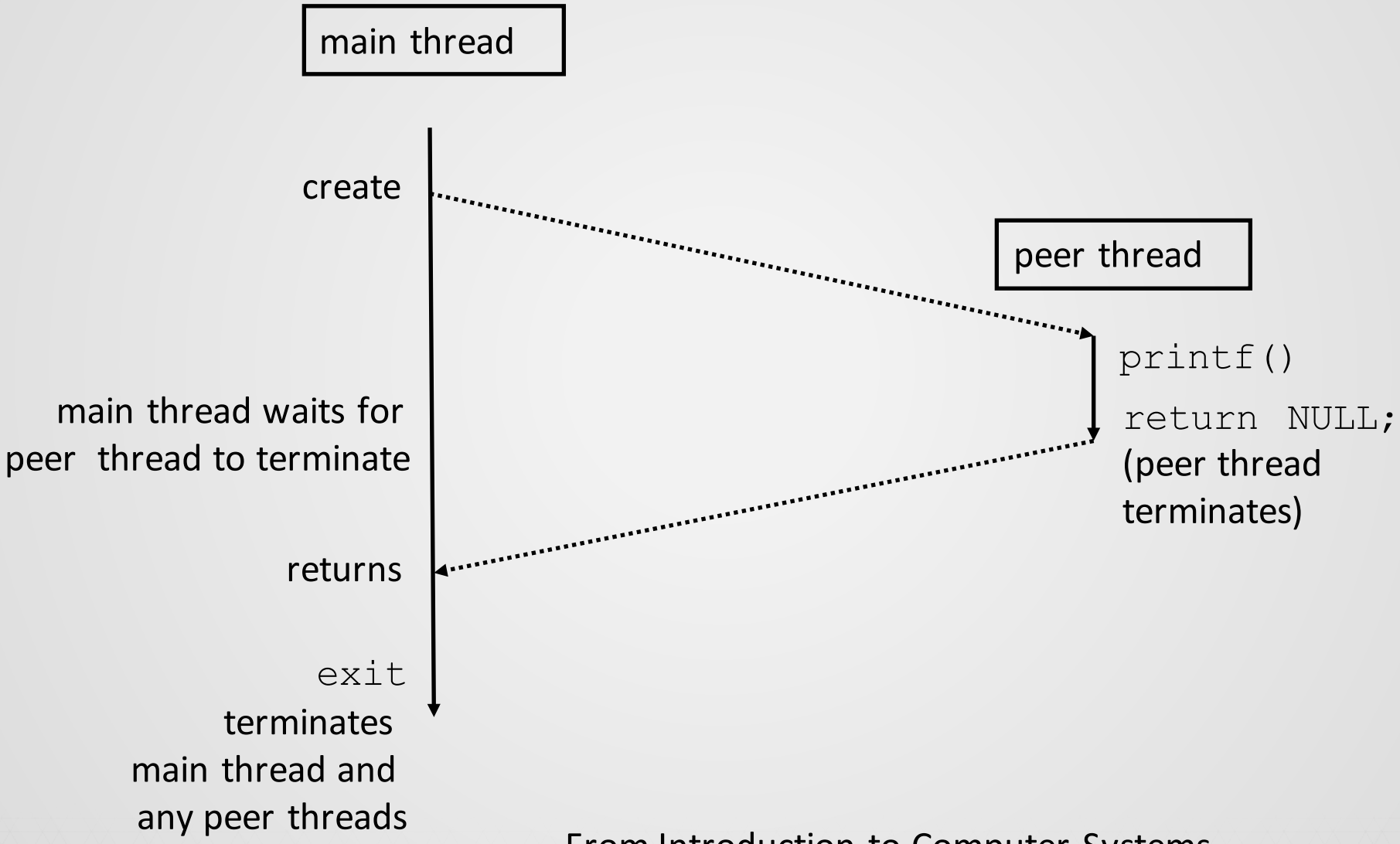
- Two threads run concurrently (are concurrent) if their logical flows overlap in time.
- Otherwise, they are sequential.

- Examples:
  - Concurrent: A & B, A&C
  - Sequential: B & C



From Introduction to Computer Systems

CONTROL FLOW WITH TWO THREADS



From Introduction to Computer Systems

THREAD UNSAFE FUNCTIONS

- Failing to protect shared variables.
- Relying on persistent state across multiple function invocations.
- Incorrect handling of pointers
- Calling a thread unsafe function

USING THREADS - OPENMP

- An API for writing portable shared memory applications with threading
  - Constructs are mostly in the form of compiler directives
    - Ignored when compiler flag for openmp is not set
  - No library to build and link
- Trivially easy to get started and thread your code
  - Often not trivial to obtain performance
  - You need to understand your decomposition and code behavior to get performance

BASICS OF OPENMP

- Constructs apply to statement blocks
  - One entry and one exit point
  - A functional call may be a part of the statement block
    - If so, necessary to make sure that function is thread-safe in the same way that the block is
- Most statement blocks can be made thread-safe by designating variables to be thread-private as needed
  - Avoid majority of race or deadlock hazards in many scientific applications

MOST USED CONSTRUCTS

- For parallelizing
  - #pragma omp parallel (!$OMP PARALLEL in FORTRAN)
  - #pragma omp for (!$OMP PARALLEL DO)
    - There are qualifiers and clauses for these constructs
- Typical use
  - Find compute intensive loops in the code
  - Make sure that there are no dependencies between different instances of the loop
  - Apply omp construct

EXAMPLE

```
#pragma omp parallel
#pragma omp for
#pragma omp parallel num_threads(4)
{ //beginning of a parallel region
   for(i=0; i<N; i++)
       a[i]=b[i]*c[i];
}// end of parallel region
```

By default the number of threads used in the loop will be the same as that defined with an environment variable. But it is possible to use fewer threads by using another construct

EXAMPLE

```
#pragma omp parallel
#pragma omp for
{ //beginning of a parallel region
    for(i=0; i<N; i++)
        a[i]=b*c[i];
}// end of parallel region
```

Still thread-safe because b is not updated, read conflicts do not cause race conditions

EXAMPLE

```
#pragma omp parallel
#pragma omp for
{ //beginning of a parallel region
   for(i=0; i<N; i++)
       a=a+b[i];
}// end of parallel region
```

Not thread-safe because a is updated, there is a write conflict which can cause race conditions

Avoid such loops when writing application from scratch. If you have them then you trade-off memory with cpu

CONCURRENT EXECUTION

- Key idea: In general, any sequentially consistent interleaving is possible, but some are incorrect!
- There are three operations: load, update, store
- Race condition, correctness depends upon threads reaching specific points in specific order
  - Assuming we start with a=0, b[i]=i+1

a=a+b[i];

i (thread)	$instr_i$	a
0	L_0	0
0	U_0	0
0	S_0	1
1	L_1	1
1	U_1	1
1	S_1	3

i (thread)	$instr_i$	a
0	L_0	0
0	U_0	0
1	L_1	0
1	U_1	0
0	S_0	1
1	S_1	2

EXAMPLE

```
#pragma omp parallel
#pragma omp for reduction (+:a)
{ //beginning of a parallel region
   for(i=0; i<N; i++)
      a=a+b[i]*c[i];
}// end of parallel region
```

EXAMPLE

```
#pragma omp parallel private(j)
#pragma omp for
{ //beginning of a parallel region
   for(i=0; i<N; i++)
     {
       j=b[i]*c[i];
#pragma omp critical
       a=a+j;
     }
}// end of parallel region
```

Barrier is implied at the end of parallel region

OTHER CONSTRUCTS

- #pragma omp master
  - executed only on the master thread
  - no implied barrier
- #pragma omp barrier
- #pragma omp single
  - executes on a single thread
  - implied barrier at the end of parallel region
- #pragma ordered
- #pragma omp sections
  - set of structured blocks to distributed among threads
- #pragma omp task
- #pragma omp threadprivate

http://openmp.org/mp-documents/OpenMP3.1-CCard.pdf

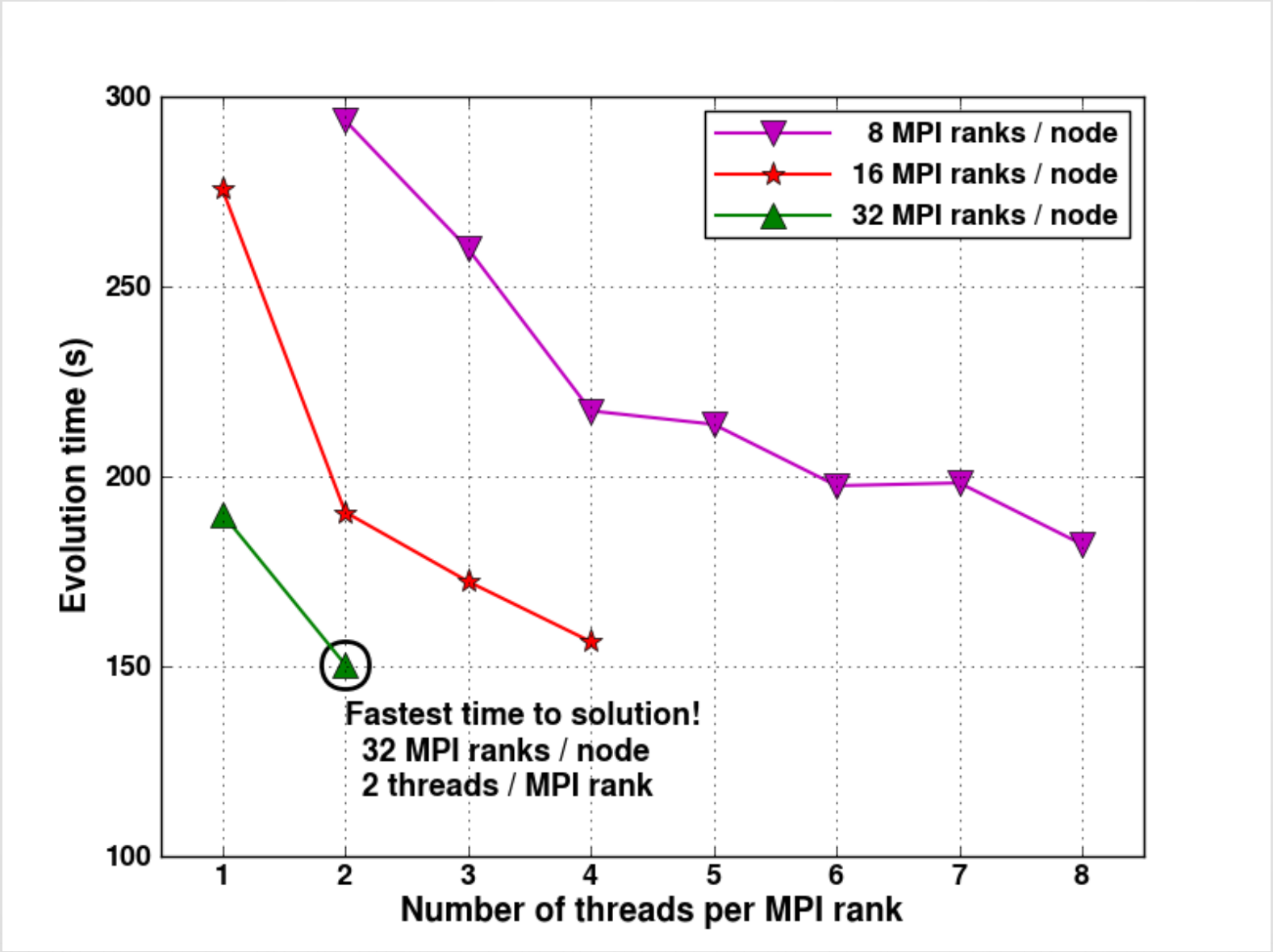
USING OPENMP WITH MIRA

- XL compilers support for OpenMP v3.1
- GCC 4.4.4 compilers support OpenMP v3.0
- Compiler flags

  Intel –openmp, GNU –fopenmp, BlueGene bgxlc_r –qsmp=omp

  -qthreaded
- Mixed mode threading is permitted
- SMP not NUMA
- Hardware threads share L1 cache – 16KB, can be a challenge

POSSIBLE CONFIGURATIONS

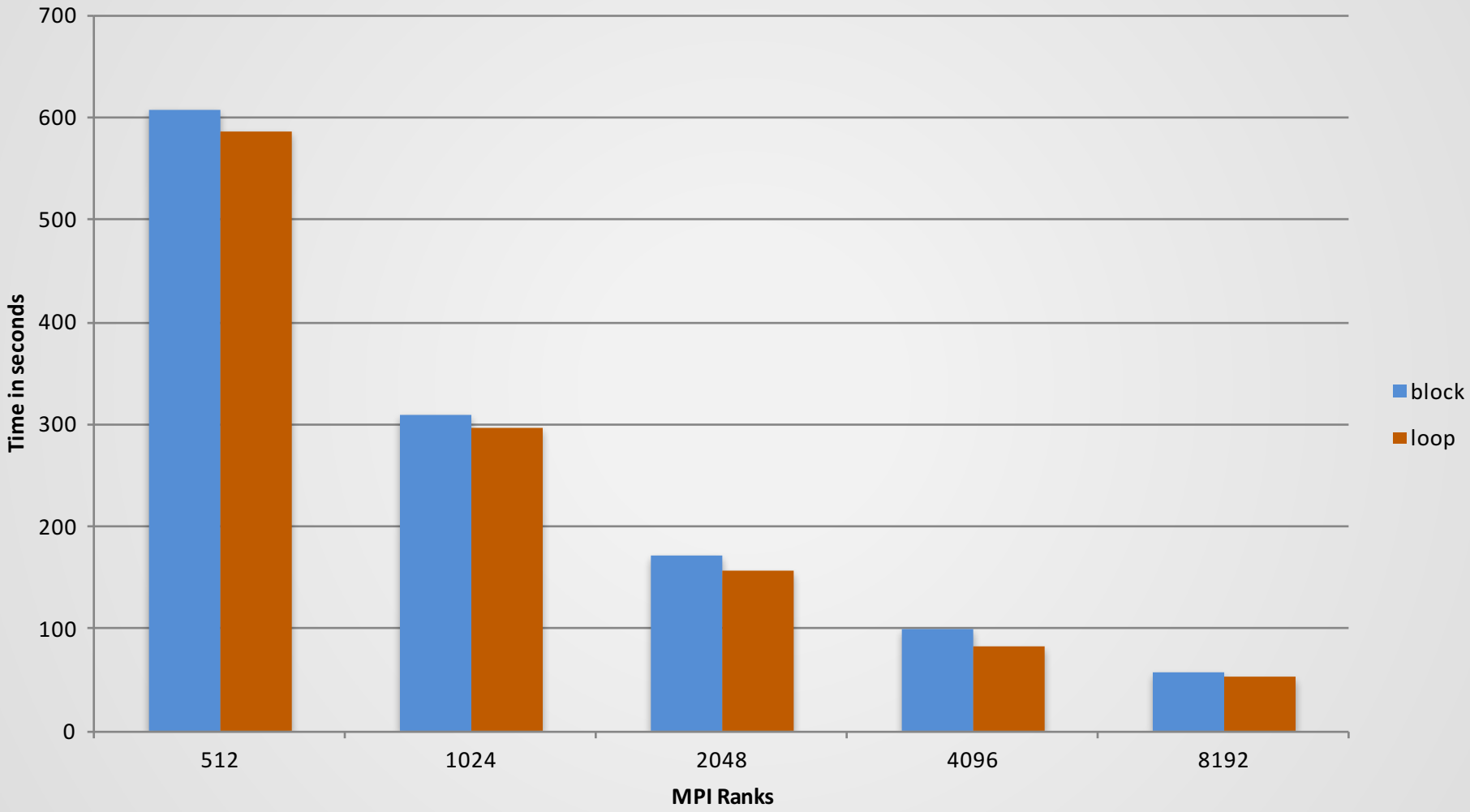


TRADE-OFFS

```
do m = 1, nc
      !$OMP PARALLEL DO PRIVATE(i,j,k)
      do k = lo(3),hi(3)
        do j = lo(2),hi(2)
          do i = lo(1),hi(1)
            up(i,j,k,m) = OneThird *   up(i,j,k,m) + &
                TwoThirds * (unp(i,j,k,m) + dt * (dp(i,j,k,m) + fp(i,j,k,m)))
          end do
        end do
      end do
      !$OMP END PARALLEL DO
    end do
  end do
```

```
#pragma omp parallel
#pragma omp for
#endif
  for(int igrid = 0; igrid < dit.size(); igrid++)
    {
      FArrayBox& bigData = a_data[dit[igrid]];
      const Box& ghostBlock = a_data[dit[igrid]].box();
FORT_SIMULATION_BLOCK(CHF_FRA(bigData),
CHF_BOX(ghostBlock),
CHF_CONST_REALVECT(a_dx));}}
```

TRADE-OFFS


700
600
500
400
300
200
100
0
Time in seconds
512
1024
2048
4096
8192
MPI Ranks
block
loop

```
!$omp parallel if (hy_threadWithinBlock) &
 !$omp default(none) &
 !$omp shared(i0,j0,k0,imax,jmax,kHydro,kUSM,kmax,k2,k3,&
 !$omp scrchFaceXPtr,scrchFaceYPtr,scrchFaceZPtr,&
 !$omp hy_fullSpecMsFluxHandling,normalFieldUpdateOnly,&
 !$omp blockID,blkLimitsGC,dt,del,ogravX,ogravY,ogravZ,FlatCoeff,hy_SpcR,hy_SpcL,hy_SpcSig,&
 !$omp hy_useGravity,U,Bx,By,Bz,hy_useGravHalfUpdate,transOrder3d,leftEig,rghtEig,lambda,&
 !$omp hy_order,hy_flattening,FlatTilde,blkLimits,hdt,hy_shockDetectOn,&
 !$omp hy_use3dFullCTU,hy_geometry,hy_useHybridOrder,hy_cfl_original,&
 !$omp hy_cfl,minCfl,&
 !$omp xCenter,DivU,kGrav,sig,hy_forceHydroLimit,hy_killdivb,datasize) &
 !$omp private(i,j,k,lbx,ubx,lby,uby,lbz,ubz,iDim, un,cf,Vc,order,Wn,Wp,sigmptr,sigcptr,sigpptr,&
 !$omp k4,im2,ip2,jm2,jp2,km2,kp2,&
 !$omp dp1,dp2,dv1,presL,presR,Sp,magPhi,magZ,sGeo_magp,sGeo_magz,&
 !$omp TransX_updateOnly,TransY_updateOnly,TransZ_updateOnly,&
 !$omp cellCfl,lowerCflAtBdry,&
 !$omp dt2dxdy6,dt2dydz6,dt2dzdx6,TransFluxXY,TransFluxYZ,TransFluxZX,&
 !$omp TransFluxYX,TransFluxZY,TransFluxXZ,Rinv,velPhi,velTht,&
 !$omp sGeo_dens,sGeo_velx,sGeo_velp,sGeo_pres,sGeo_trans,sGeo_eint,&
 !$omp geoFac,cs,eta,enth,dir,HY_velPhi,HY_velTht,h_magphi,h_magz)
```

USEFUL LINKS

OpenMP site : http://openmp.org

Quick reference : http://openmp.org/mp-documents/OpenMP3.1-CCard.pdf

Introductory material: http://openmp.org/mp-documents/omp-hands-on-SC08.pdf

Comprehensive tutorial:
https://computing.llnl.gov/tutorials/openMP

MIRA specific details:
http://www.alcf.anl.gov/user-guides/how-manage-threading#specification