Bring your application to a new era:

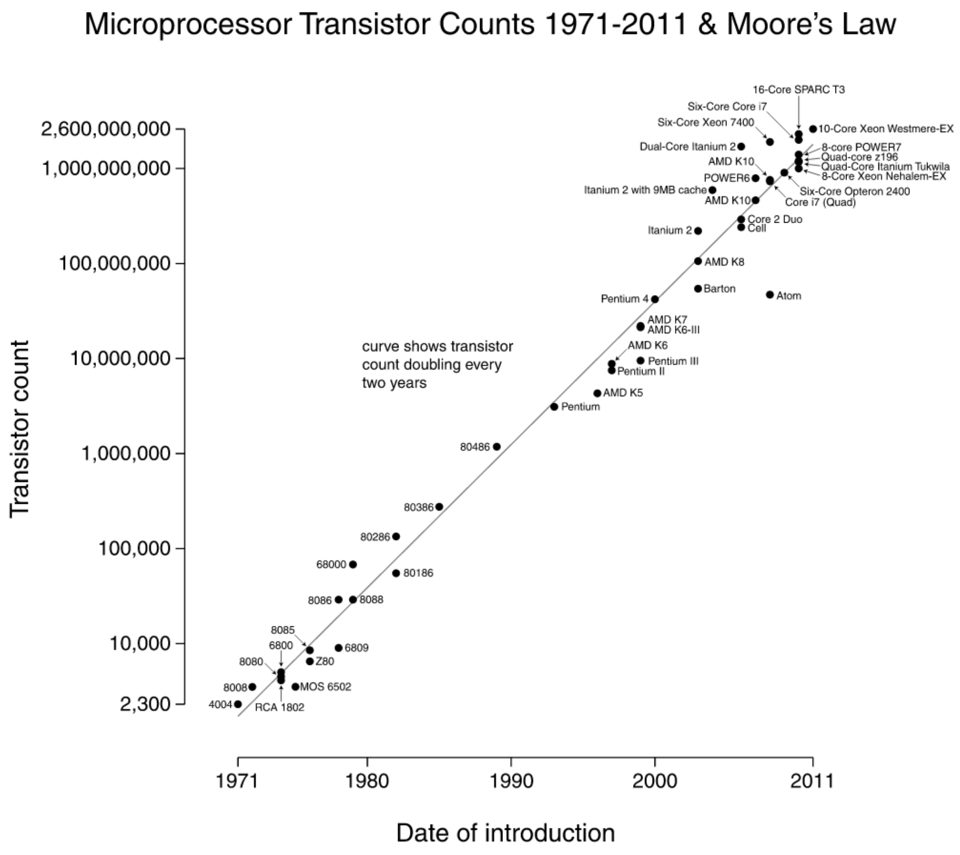
learning by example how to parallelize and optimize for Intel® Xeon® processor and Intel® Xeon Phi™ coprocessor

Manel Fernández, Roger Philp, Richard Paul

Bayncore Ltd.

HPCKP'15

Moore's law: how to use so many transistors?



Single thread performance is limited

- Clock frequency constraints
- "Near" parallelism harder to expose
  - Instruction level parallelism (ILP)

Hint: exploit "distant" parallelism

- Data level parallelism (DLP)
- Task level parallelism (TLP)

Programmers responsibility to expose DLP/TLP parallelism

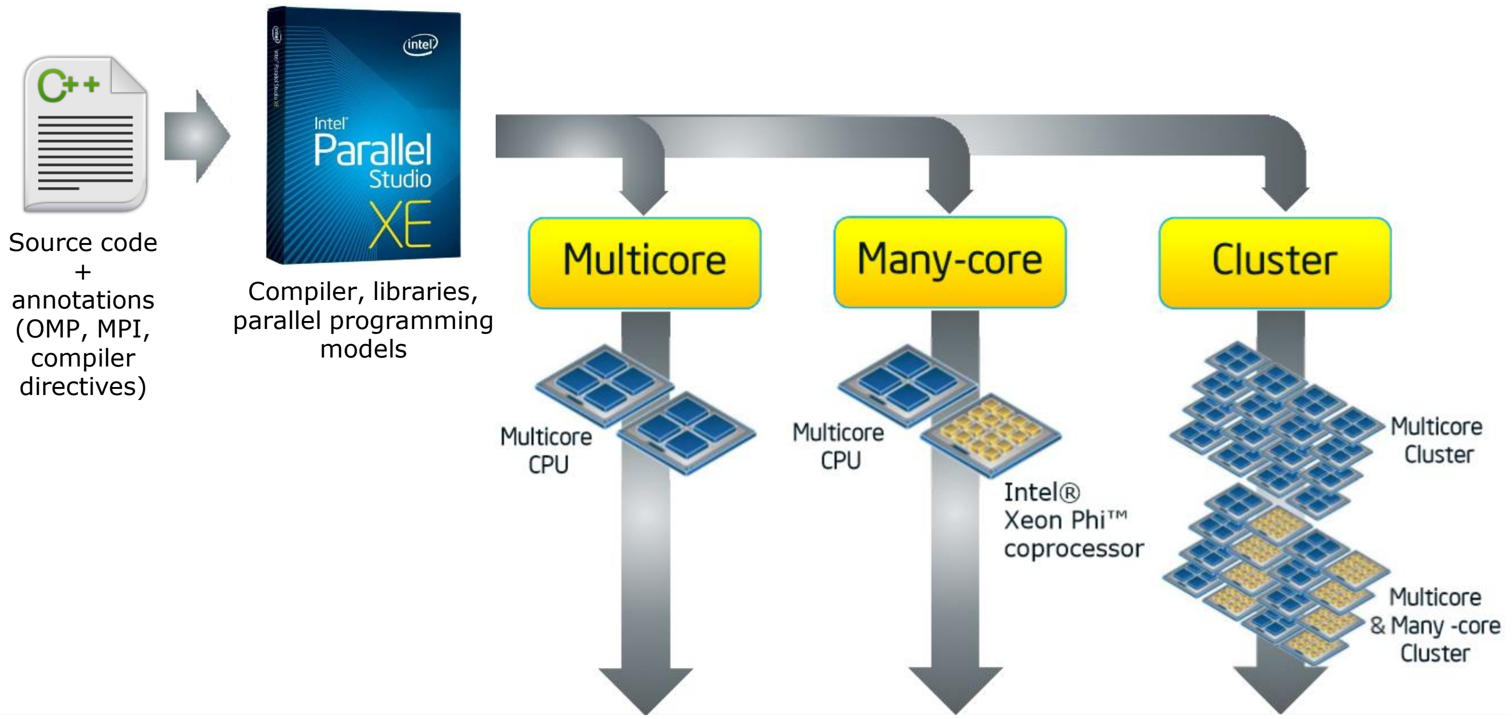
"Transistor Count and Moore's Law - 2011" by Wgsimon - http://en.wikipedia.org/wiki/Moore's_law

The multi- and many-core era: Intel® solutions for HPC

Multi-core	Many integrated core (MIC)
C/C++/Fortran, OMP/MPI/Cilk+/TBB	C/C++/Fortran, OMP/MPI/Cilk+/TBB
Bootable, native execution model	PCIe, native and offload execution models
Up to 18 cores, 3 GHz, 36 threads	Up to 61 cores, 1.2 GHz, 244 threads
Up to 768 GB, 68 GB/s, 432 GFLOP/s DP	Up to 16 GB, 352 GB/s, 1.2 TFLOP/s DP
256-bit SIMD, FMA, gather (AVX2)	512-bit SIMD, FMA, gather/scatter, EMU (IMCI)
Targeted at general purpose applications Single thread performance (ILP) Memory capacity	**Targeted at highly parallel applications** High parallelism (DLP, TLP) High memory bandwidth

How to enable parallelism with standard methods

Intel® Parallel Studio XE 2015 tool suite



Single programming model for all your code

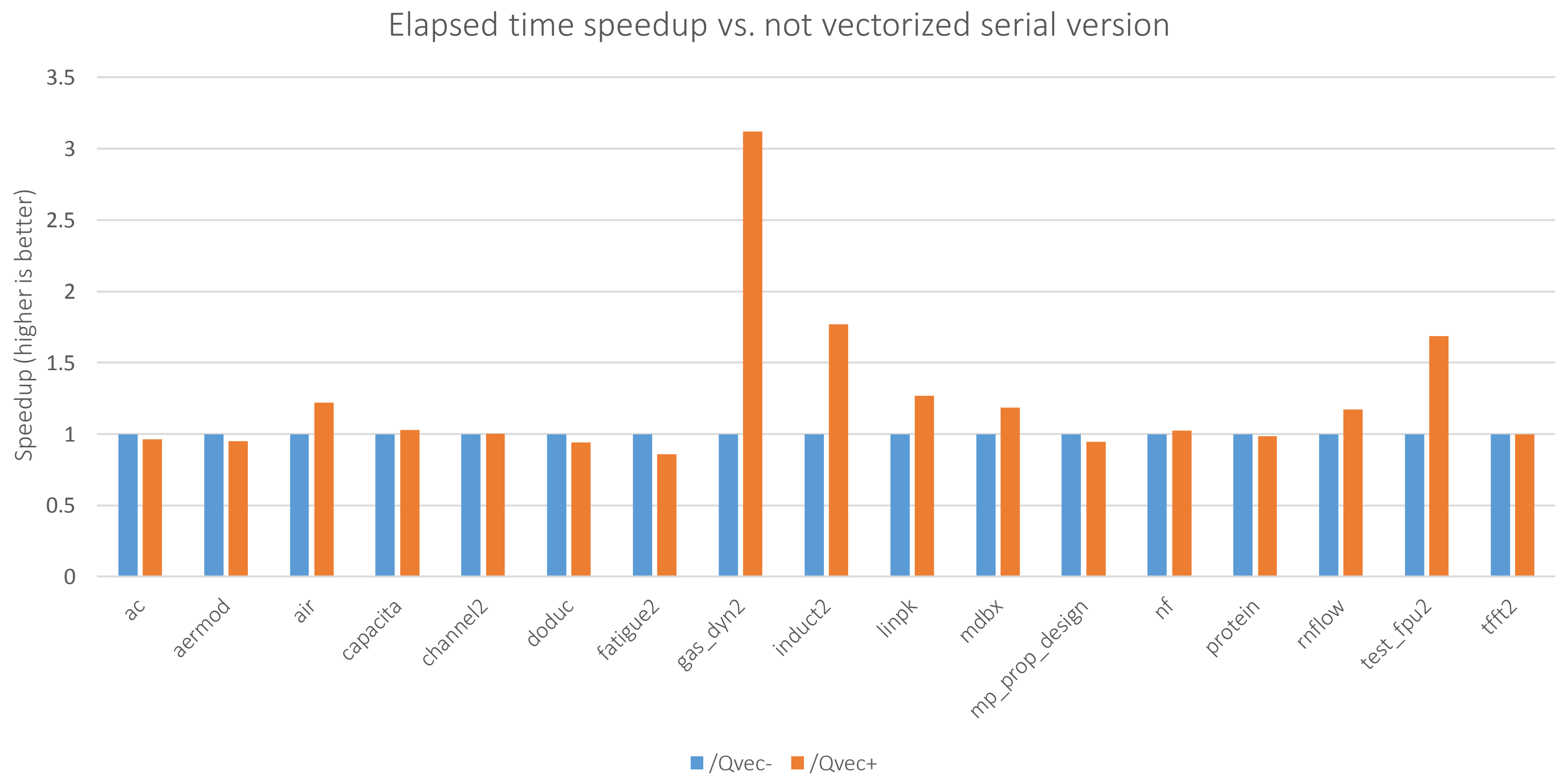
Characterizing Polyhedron benchmark suite

Windows 8

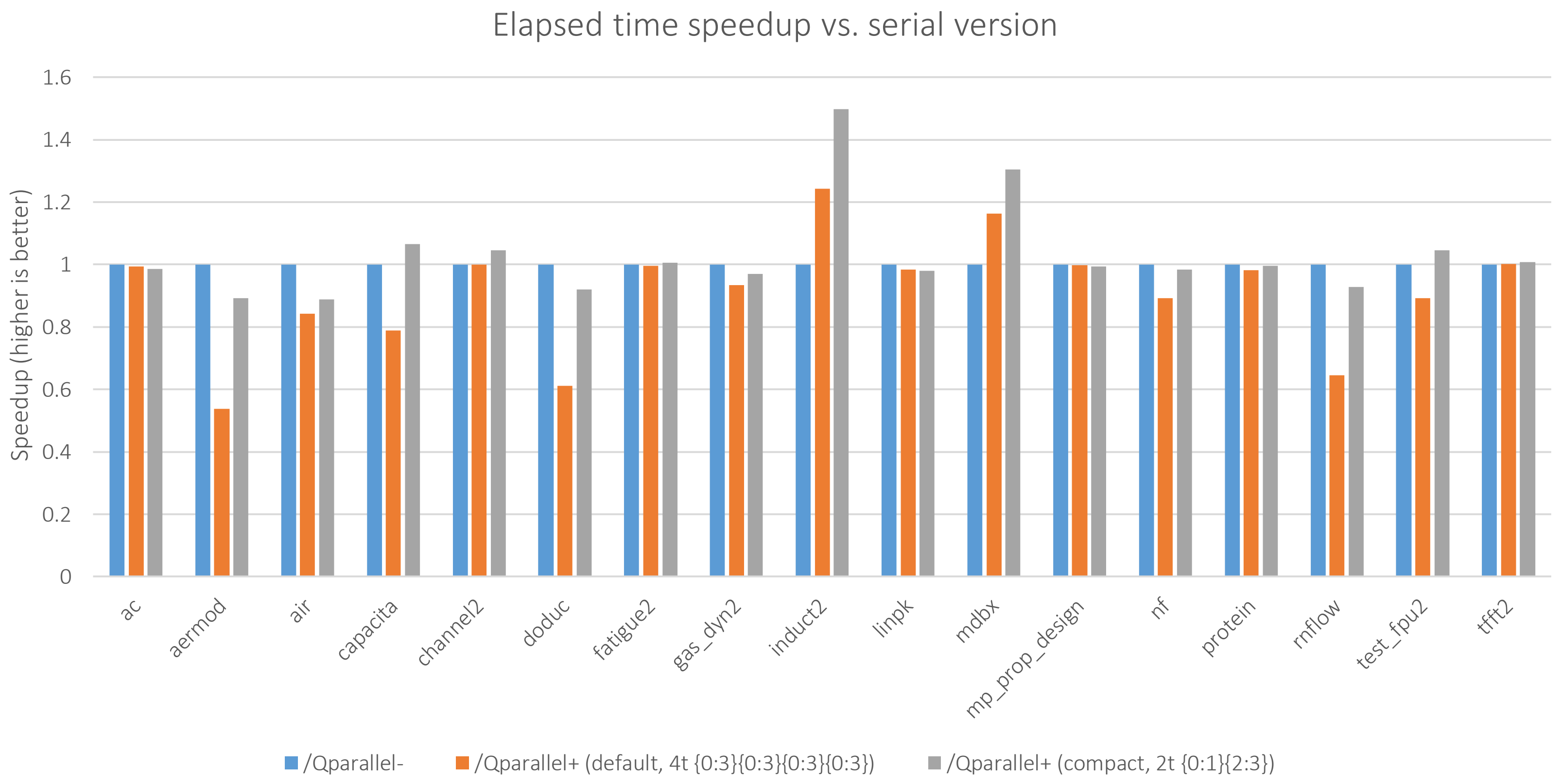
Intel® Core™ i7-4500U (0,1)(2,3)

Intel® Fortran Compiler 15.0.1.14 [/O3 /fp:fast=2 /align:array64byte /Qipo /QxHost]

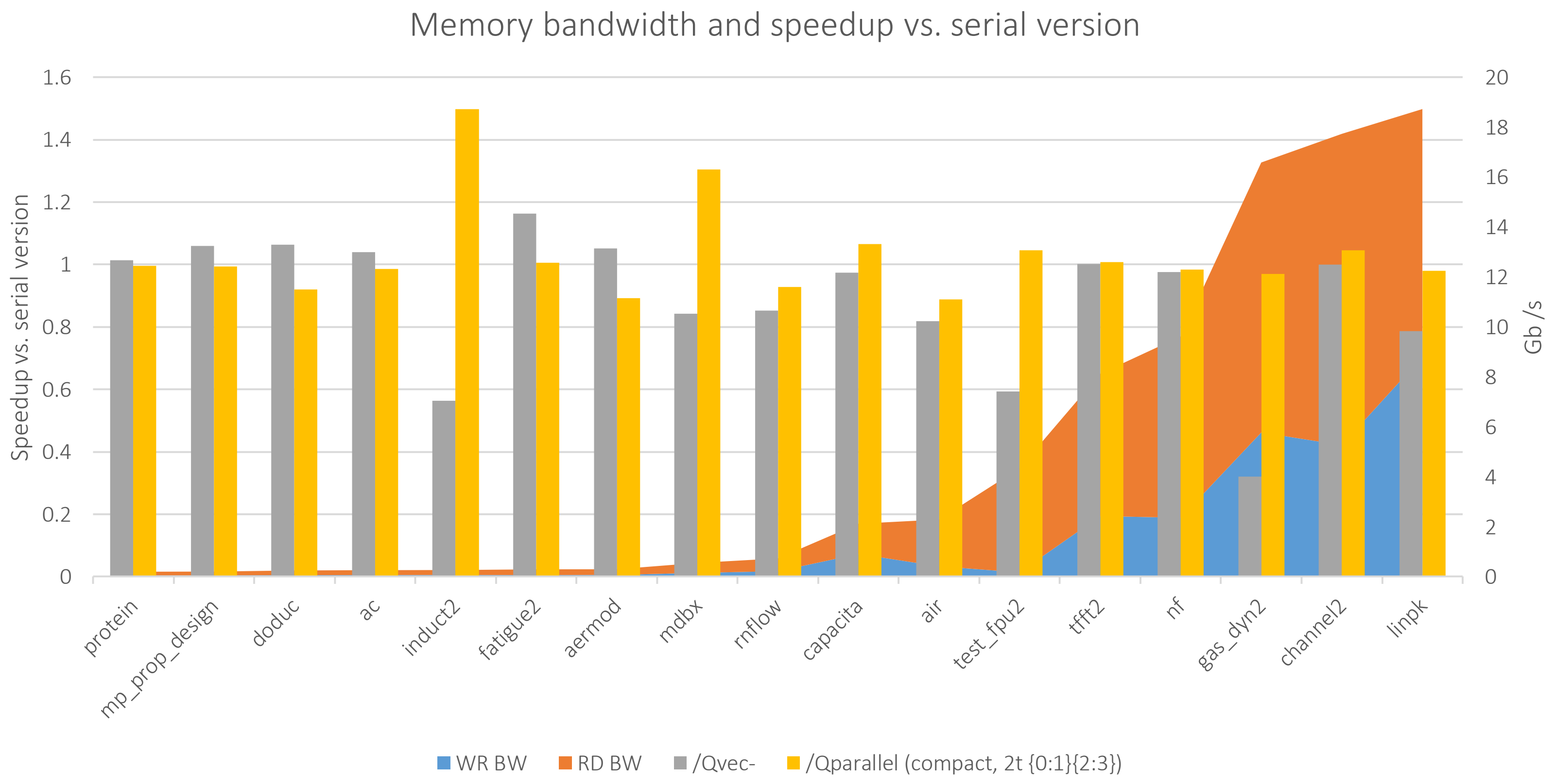
Auto-vectorization effectiveness



Auto-parallelization effectiveness



Memory bandwidth requirements



Observations: implicit vs. explicit parallelism

Compiler toolchain is limited in exposing implicit parallelism

- Good for ILP (uArch supposed to help)
- Not so bad for DLP
  - Exploited by use of “vectors” (SIMD)
  - But potentially missing opportunities due to aliasing, etc.
- Disappointing for TLP
  - Hyper-threading rarely useful on HPC applications

Explicit parallelism relies on the programmer

- DLP: compiler directives, array notation, vector classes, intrincsics
- TLP: Multi- and many-cores available (OpenMP, Cilk+, TBB)

Distributed systems with standard methods

- Clusters, MPI models

BAYNCORE

Exposing DLP/TLP parallelism

Simplest method by using compiler directives (aka “pragmas”)

Exposing DLP: vectorization/SIMD pragmas

`#pragma vector {args}`	Vectorization hints
`#pragma ivdep`	Ignore vector assumed dependencies
`#pragma simd [clauses]`	Enforces vectorization with hints

Exposing TLP: OMP pragmas

`#omp parallel for`	Parallelizes iterations of a given loop
`#omp atomic/critical`	Thread synchronization

Runtime performance tuning for threaded applications

`OMP_NUM_THREADS`	Number of threads to run
`OMP_SCHEDULE`	How work is distributed among threads
`KMP_AFFINITY`	How threads are bound to physical PUs
`KMP_PLACE_THREADS`	Easy thread placement (Intel® Xeon Phi™ only)

BAYNCORE

Polyhedron/gas_dyn2

Linux RHEL 6.6

Intel® Xeon® E5-4650L, 2 socket x 8 cores x 2 HTs

Intel® Xeon Phi™ 7120A, 61 cores x 4 threads

Intel® Fortran Compiler 15.0.1.14 [-O3 -fp-model fast=2 -align array64byte -ipo -xHost/-mmic]

Serial version

Continuity equations solver to models the flow of a gas in 1D

Two main hotspots: EOS (66%) and CHOZDT(33%)

- Implicit loops by using Fortran 90 array notation
- Both hotspots perfectly fused + vectorized

```
SUBROUTINE EOS(NODES, IENER, DENS, PRES, TEMP, GAMMA, CS, SHEAT, CGAMMA)
!-----------------------------------------------
      INTEGER NODES
      REAL SHEAT, CGAMMA
      REAL, DIMENSION(NODES) :: IENER, DENS, PRES, TEMP, GAMMA, CS
!-----------------------------------------------
      TEMP(:NODES) = IENER(:NODES)/SHEAT
      PRES(:NODES) = (CGAMMA - 1.0)*DENS(:NODES)*IENER(:NODES)
      GAMMA(:NODES) = CGAMMA
      CS(:NODES) = SQRT(CGAMMA*PRES(:NODES)/DENS(:NODES))
```

```
SUBROUTINE CHOZDT(NODES, VEL, SOUND, DX, DT)
!-----------------------------------------------
      INTEGER NODES, ISET(1)
      REAL, DIMENSION (NODES) :: VEL, DX, SOUND, DTEMP
!-----------------------------------------------
      DTEMP = DX/(ABS(VEL) + SOUND)
      ISET = MINLOC (DTEMP)
```

OMP workshare construct

Workshare currently not working (not parallelized)

Reduction loop in CHOZDT does not even vectorize

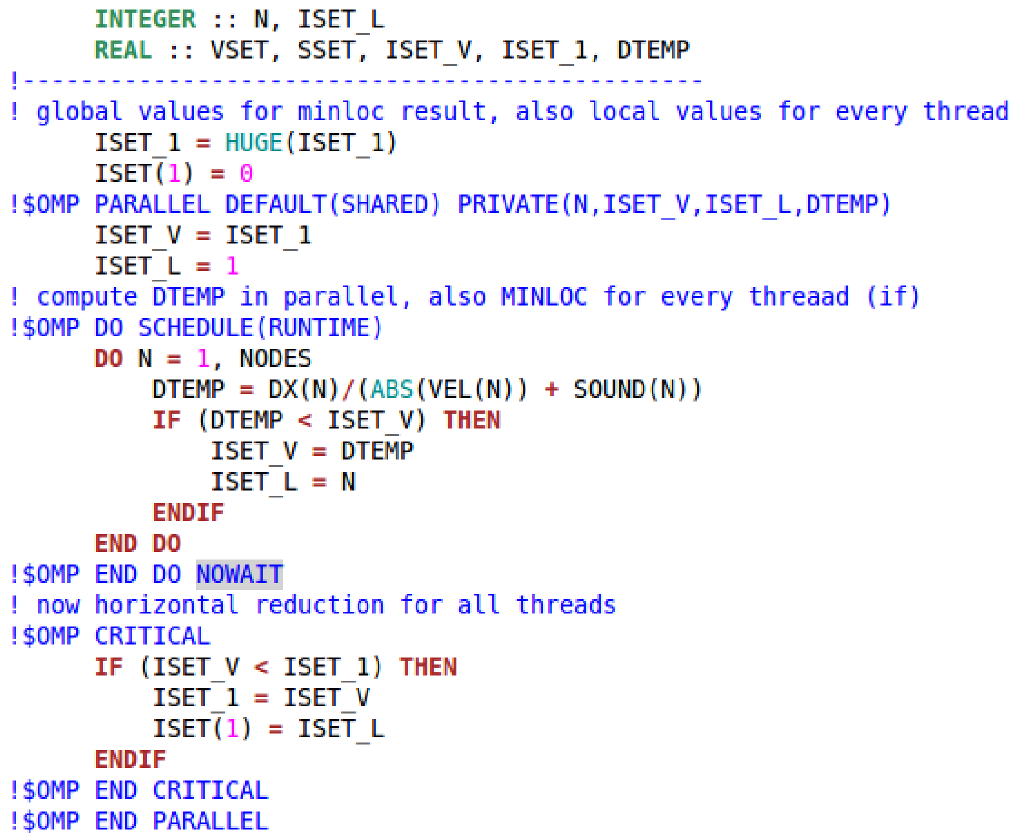
```
!$OMP PARALLEL WORKSHARE DEFAULT(SHARED)
      TEMP(:NODES) = IENER(:NODES)/SHEAT
      PRES(:NODES) = (CGAMMA - 1.0)*DENS(:NODES)*IENER(:NODES)
      GAMMA(:NODES) = CGAMMA
      CS(:NODES) = SQRT(CGAMMA*PRES(:NODES)/DENS(:NODES))
!$OMP END PARALLEL WORKSHARE
```

```
!$OMP PARALLEL WORKSHARE DEFAULT(SHARED)
      DTEMP = DX/(ABS(VEL) + SOUND)
      ISET = MINLOC (DTEMP)
!$OMP END PARALLEL WORKSHARE
```

OMP parallel loop (CHOZDT)

Intel® compiler does not support OMP 4.0 user defined reductions

We have to write the parallel reduction by ourselves!



```
      INTEGER :: N, ISET_L
      REAL :: VSET, SSET, ISET_V, ISET_1, DTEMP
!-----------------------------------------------
! global values for minloc result, also local values for every thread
      ISET_1 = HUGE(ISET_1)
      ISET(1) = 0
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(N,ISET_V,ISET_L,DTEMP)
      ISET_V = ISET_1
      ISET_L = 1
! compute DTEMP in parallel, also MINLOC for every threaad (if)
!$OMP DO SCHEDULE(RUNTIME)
      DO N = 1, NODES
          DTEMP = DX(N)/(ABS(VEL(N)) + SOUND(N))
          IF (DTEMP < ISET_V) THEN
              ISET_V = DTEMP
              ISET_L = N
          ENDIF
      END DO
!$OMP END DO NOWAIT
! now horizontal reduction for all threads
!$OMP CRITICAL
      IF (ISET_V < ISET_1) THEN
          ISET_1 = ISET_V
          ISET(1) = ISET_L
      ENDIF
!$OMP END CRITICAL
!$OMP END PARALLEL
```

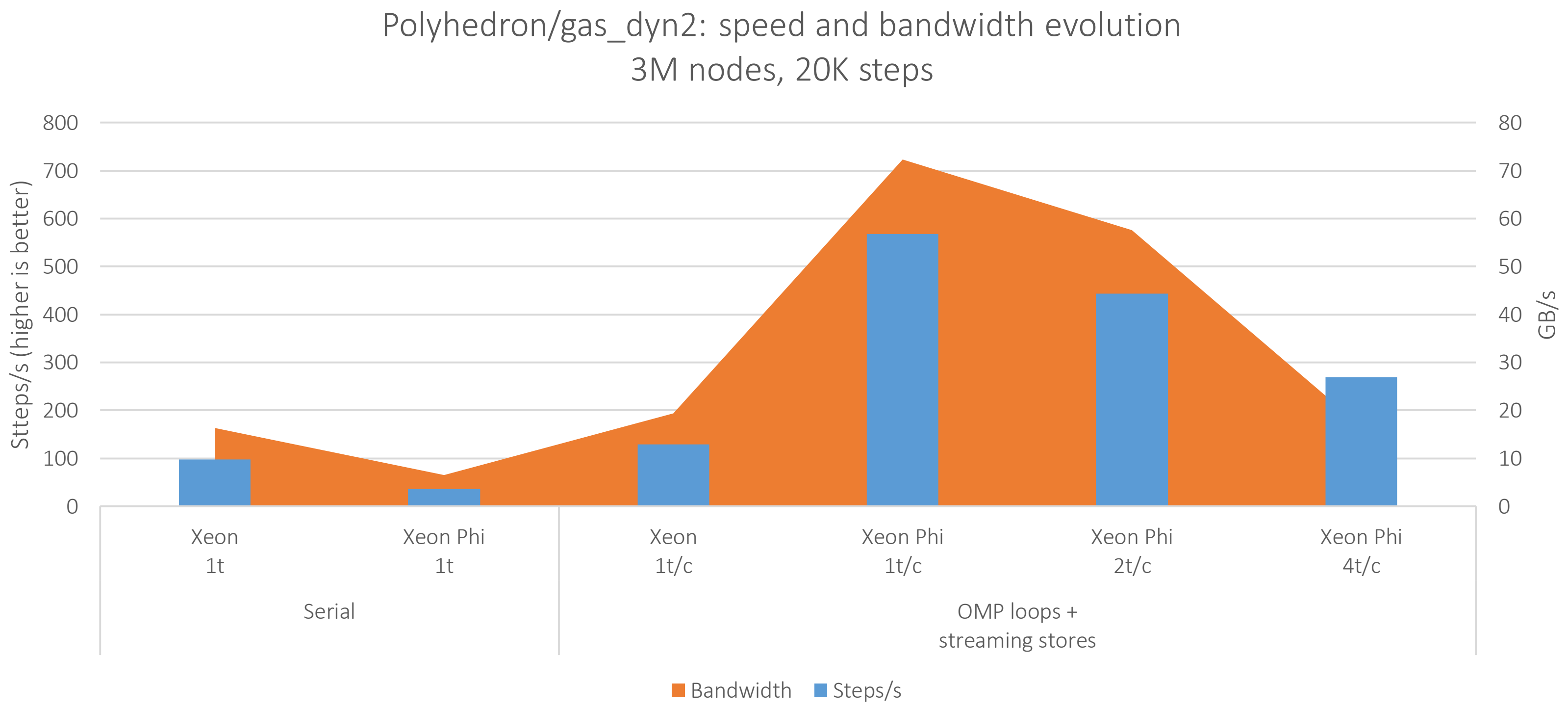
OMP parallel loop (EOS)

Straightforward transformation

Streaming stores to avoid wasting some read bandwidth

```
!$OMP PARALLEL DO DEFAULT(SHARED) PRIVATE(N) SCHEDULE(RUNTIME)
!DIR$ VECTOR NONTEMPORAL(TEMP,PRES,GAMMA,CS)
      DO N = 1, NODES
          TEMP(N) = IENER(N)/SHEAT
          PRES(N) = (CGAMMA - 1.0)*DENS(N)*IENER(N)
          GAMMA(N) = CGAMMA
          CS(N) = SQRT(CGAMMA*PRES(N)/DENS(N))
      END DO
!$OMP END PARALLEL DO
```

Performance results



Intel® Xeon Phi^{TM} speedup vs. Intel® Xeon®: 5.8x (serial), 4.4x (parallel)

Polyhedron/linpk

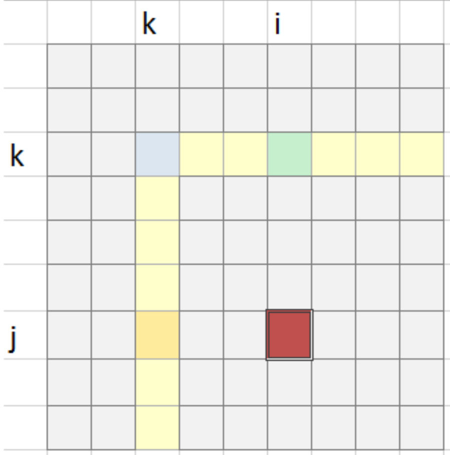
Linux RHEL 6.6

Intel® Xeon® E5-4650L, 2 socket x 8 cores x 2 HTs

Intel® Xeon Phi™ 7120A, 61 cores x 4 threads

Intel Fortran Compiler 15.0.1.14 [-O3 -fp-model fast=2 -align array64byte -ipo -xHost/-mmic]

Linpk hotspot: DGEFA



```
DO k = 1 , N-1
   ! find l = pivot index
   ...
   ! interchange if necessary
    ...
   ! compute multipliers
   DO i = k+1, N
      A(i,k) = - A(i,k) / A(k,k)
   ENDDO
   ! row elimination with column indexing
   DO j = k+1, N
      DO i = k+1, N
(*)      A(i,j) = A(i,j) + A(i,k) * A(k,j)
      ENDDO
   ENDDO
ENDDO
```

Matrix decomposition with partial pivoting by Gaussian elimination

Invokes BLAS routines DAXPY (98%), IDAMAX, DSCAL (all are inlined)

OMP parallel loop

Inner “i” loop properly autovectorized by the compiler

Middle “j” loop can be parallelized

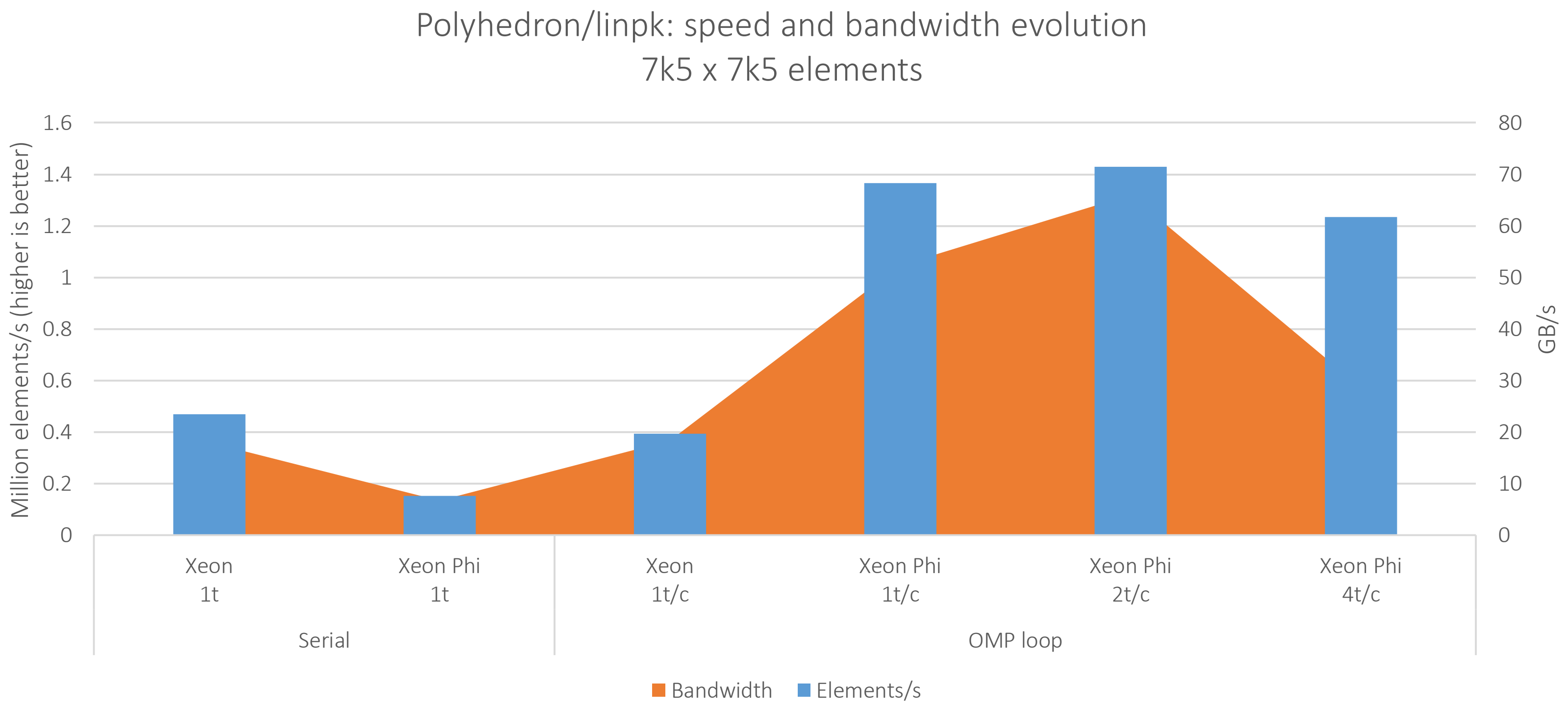
Outer “k” loop (diagonal) has dependencies between iterations

Application is memory bound

```
      SUBROUTINE DGEFA(A,Lda,N,Ipvt,Info)
! gaussian elimination with partial pivoting
      INTEGER Lda, N, Ipvt(*), Info
      DOUBLE PRECISION A(Lda,*)
      DOUBLE PRECISION t
      INTEGER IDAMAX, j, k, l

      Info = 0
      IF ( N.GT.1 ) THEN
         DO k = 1, N-1
! find l = pivot index
!           ...
! zero pivot implies this column already triangularized
            IF ( A(l,k).EQ.0.0D0 ) THEN
               Info = k
            ELSE
! interchange if necessary
!           ...
! compute multipliers
!           ...
! row elimination with column indexing
!$OMP PARALLEL DO DEFAULT(SHARED) PRIVATE(j,i) SCHEDULE(RUNTIME)
               DO j = k+1, N
                  DO i = k+1, N
                     A(i,j) = A(i,j) + A(i,k) * A(k,j)
                  ENDDO
               ENDDO
!$OMP END PARALLEL DO
            ENDIF
         ENDDO
      ENDIF
```

Performance results



Intel® Xeon Phi^{TM} speedup vs. Intel® Xeon®: 3x (serial), 3.6x (parallel)

Summary and conclusions

Programmers are responsible of exposing DLP/TLP parallelism to fully exploit the available hardware in HPC domains

Today’s Intel® HPC solutions allow to easily expose DLP/TLP parallelism

- Intel® Parallel Studio XE 2015 tool suite
- Simple methods (compiler pragmas, OMP, libraries)
- Same source code for multi- and many-core processors

Intel® Xeon Phi™ coprocessors targeted at highly parallel applications

- Significant speedups achieved in bandwidth bound applications
- Runtime tuning is key to achieve best performance

Future work

- Experiment with other benchmarks (not only from Polyhedron)
  - Non memory bound applications, native/offload execution models
- Extend parallelization to distributed systems with MPI