Optimizing Geometry Routines for (SIMD) Vector Particle Tracking

-- goals, challenges, and first results --

Sandro Wenzel / CERN-PH-SFT

in collaboration with F. Carminati, P. Mato, J. Apostolakis

Concurrency Forum 05.06.2013

Motivation

- Today, many code optimization efforts target **multicore/threading**
  - For example in simulation software: looking forward to **multithreaded Geant4** (Geant4-MT) coming in the next release

Motivation

- Today, many code optimization efforts target **multicore/threading**
  - For example in simulation software: looking forward to **multithreaded Geant4** (Geant4-MT) coming in the next release
- to exploit full potential will have to make use of other performance dimensions, in particular of **growing vector instruction sets**

Motivation

- Today, many code optimization efforts target **multicore/threading**
  - For example in simulation software: looking forward to **multithreaded Geant4** (Geant4-MT) coming in the next release
- to exploit full potential will have to make use of other performance dimensions, in particular of **growing vector instruction sets**
- In simulation, some efforts start(ed) thinking **beyond threading**:
  - e.g., "Geant Vector Prototype" (F. Carminati et al.)
  - GPU simulation prototype (P. Canal et al., Fermilab)

Motivation

- Today, many code optimization efforts target **multicore/threading**
  - For example in simulation software: looking forward to **multithreaded Geant4** (Geant4-MT) coming in the next release
- to exploit full potential will have to make use of other performance dimensions, in particular of **growing vector instruction sets**
- In simulation, some efforts start(ed) thinking **beyond threading**:
  - e.g., "Geant Vector Prototype" (F. Carminati et al.)
  - GPU simulation prototype (P. Canal et al., Fermilab)
- basic idea: from processing of **single particles** to **vectors of particles**

⇒ **data parallelism**

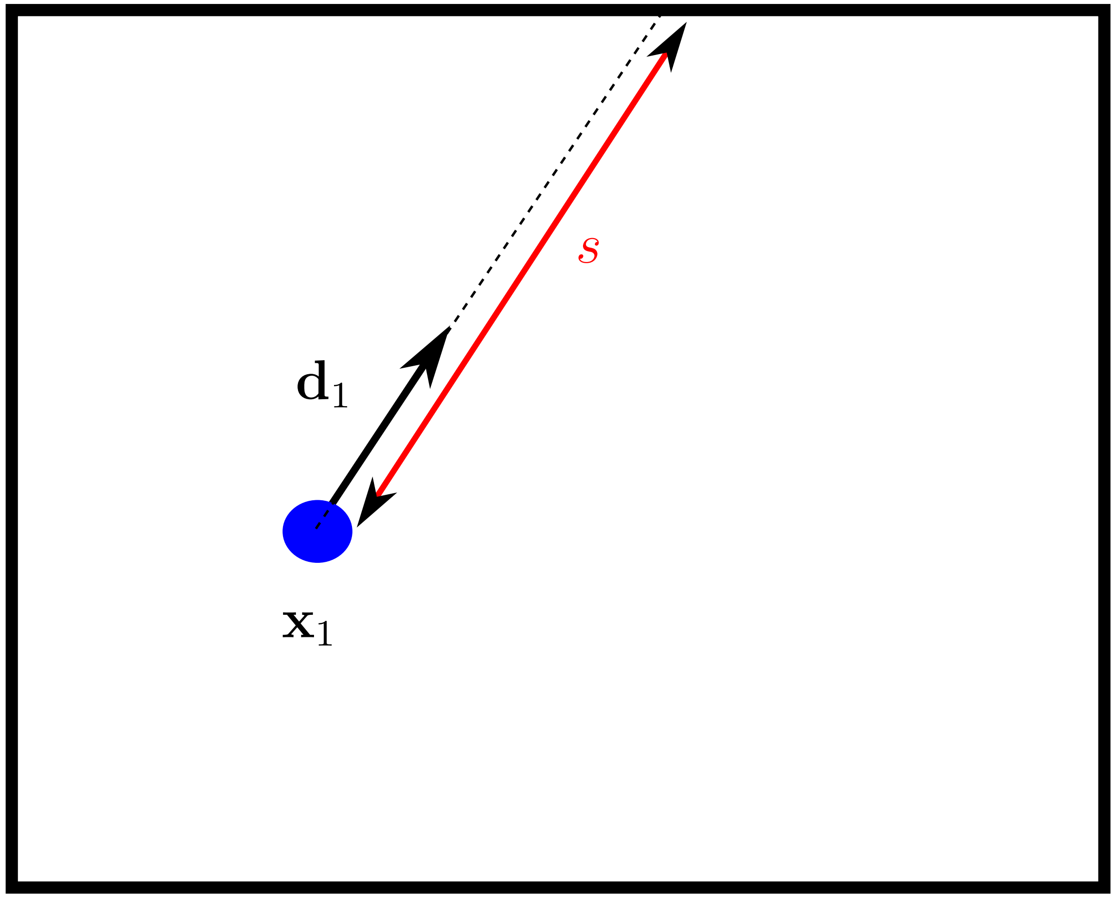
⇒ * **use of SIMD instructions, GPU**

⇒ * **benefit from less cache misses**

TGeoBBox: Example of Vector Processing

- typical geometry task in particle tracking: **get distance to boundary**

```
double
Box::DistFromInside(double *x, double *d);
```

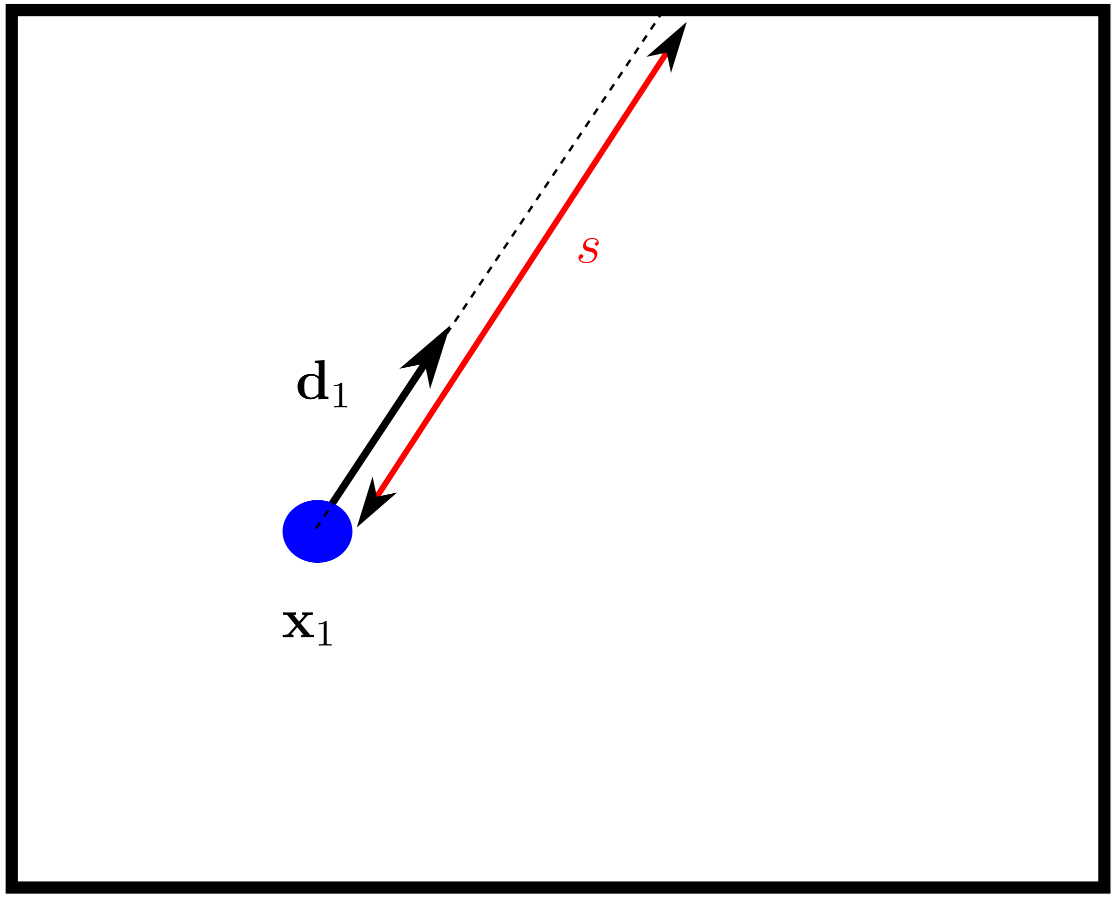


1 particle

TGeoBBox: Example of Vector Processing

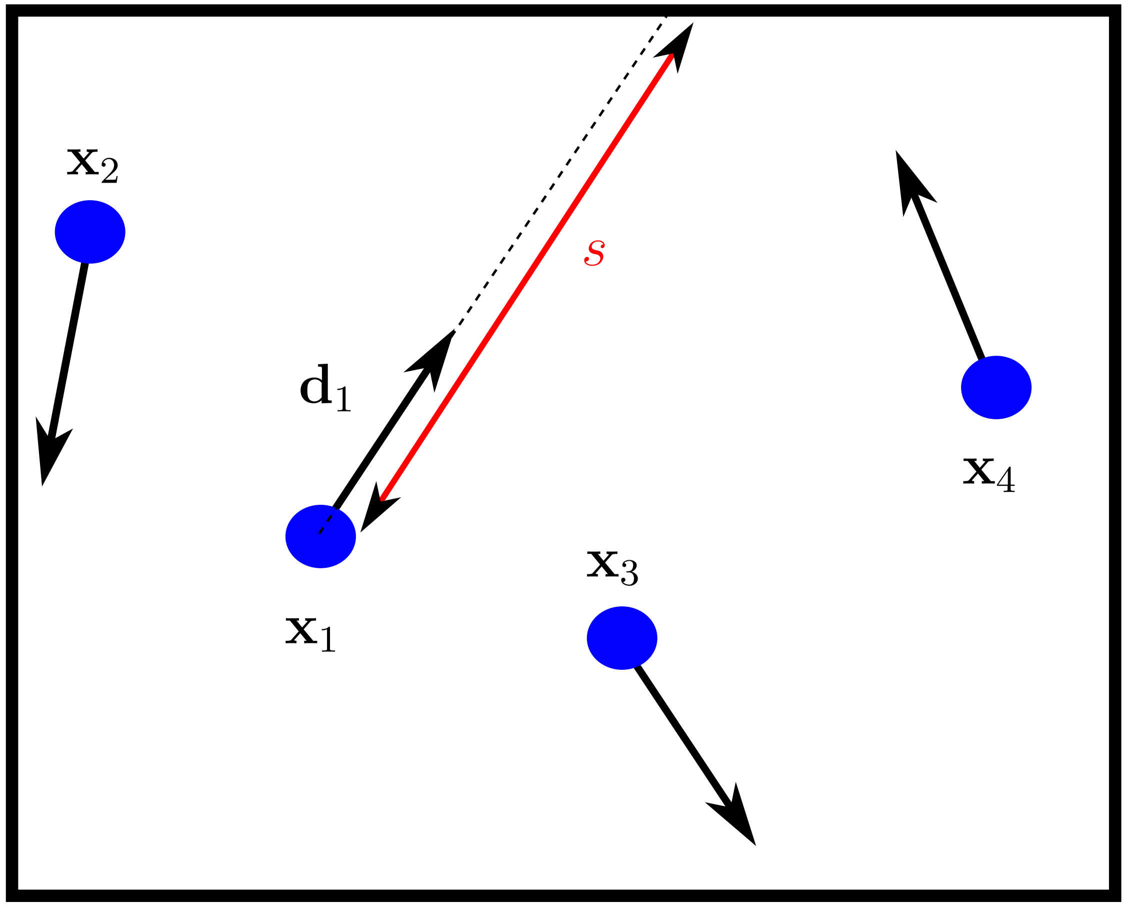
- typical geometry task in particle tracking: **get distance to boundary**

```
double
Box::DistFromInside(double *x, double *d);
```



1 particle

```
void
Box::DistFromInside_v(double *x,
double *d, double *dist, int np);
```

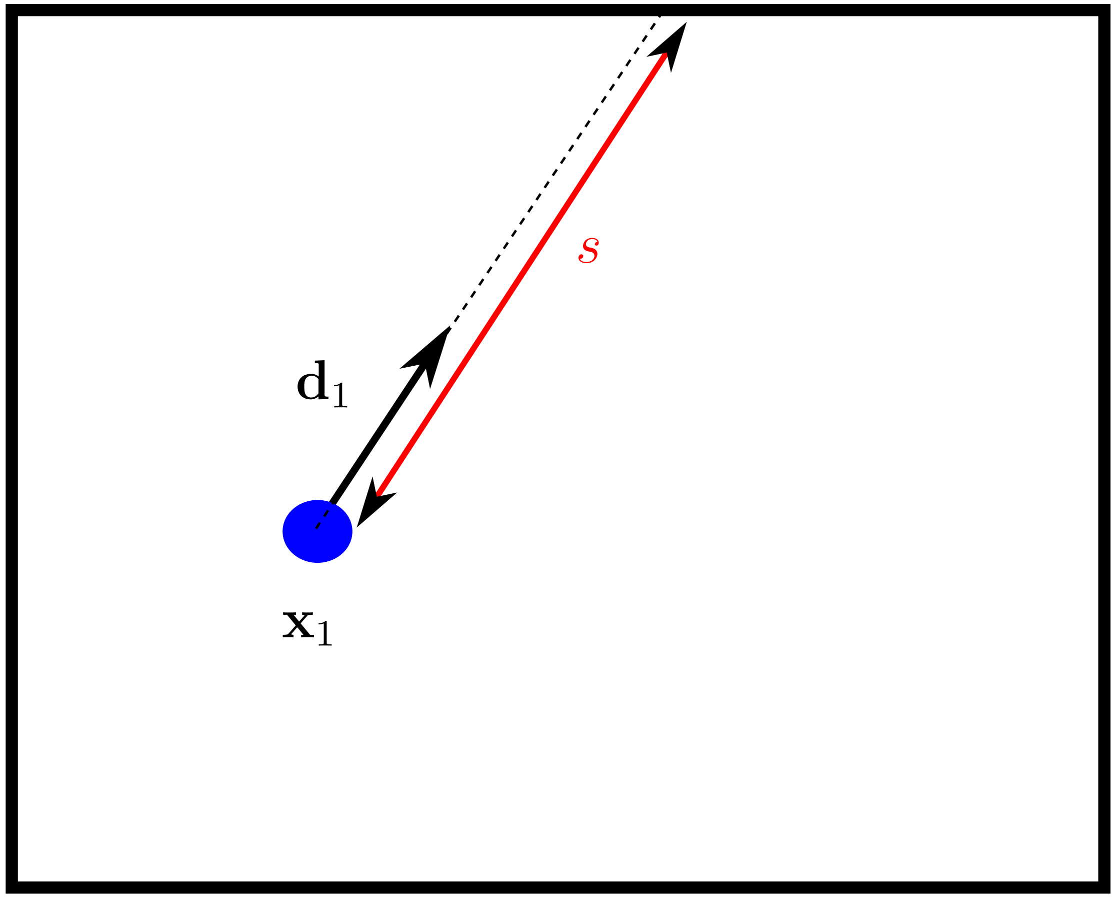


vector of particles

TGeoBBox: Example of Vector Processing

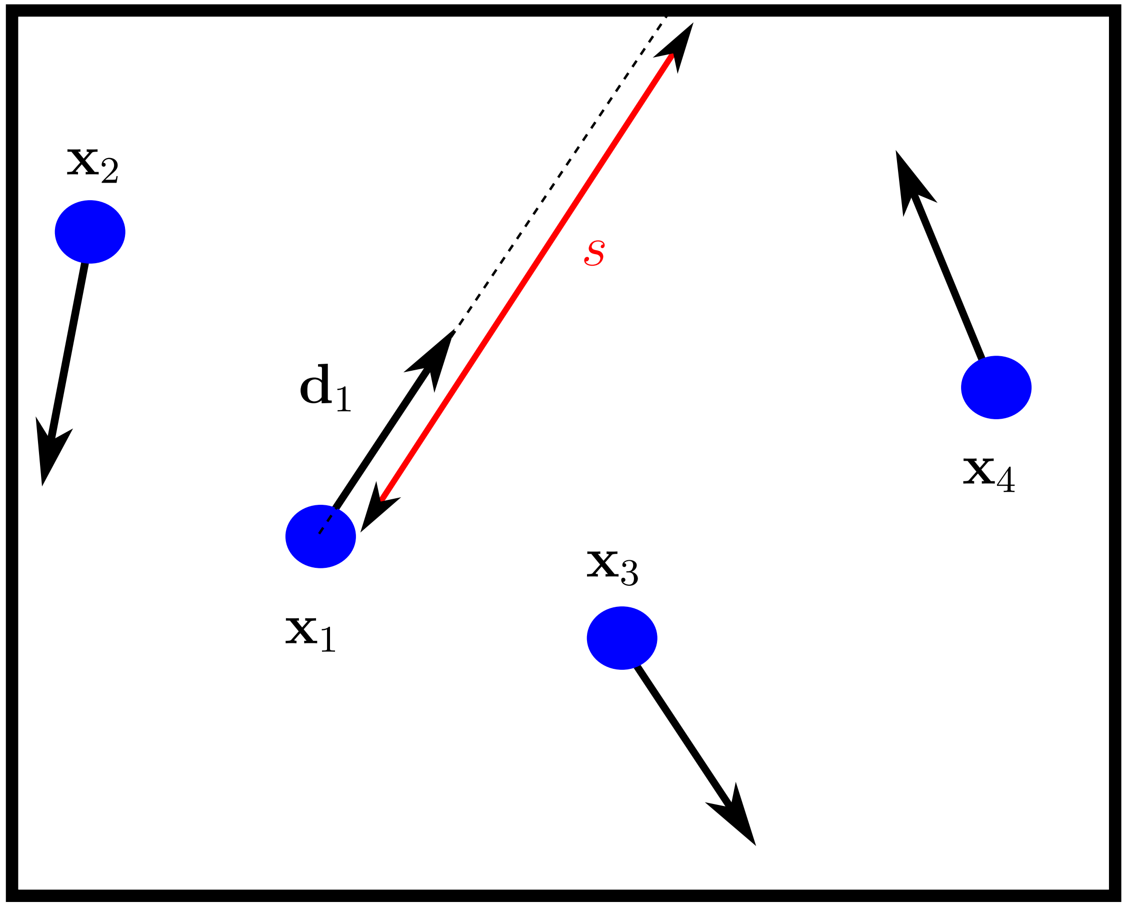
- typical geometry task in particle tracking: **get distance to boundary**

```
double
Box::DistFromInside(double *x, double *d);
```



1 particle

```
void
Box::DistFromInside_v(double *x,
double *d, double *dist, int np);
```



vector of particles

- what can we **gain** from this, assuming that we manage to have vector of particles in the first place?

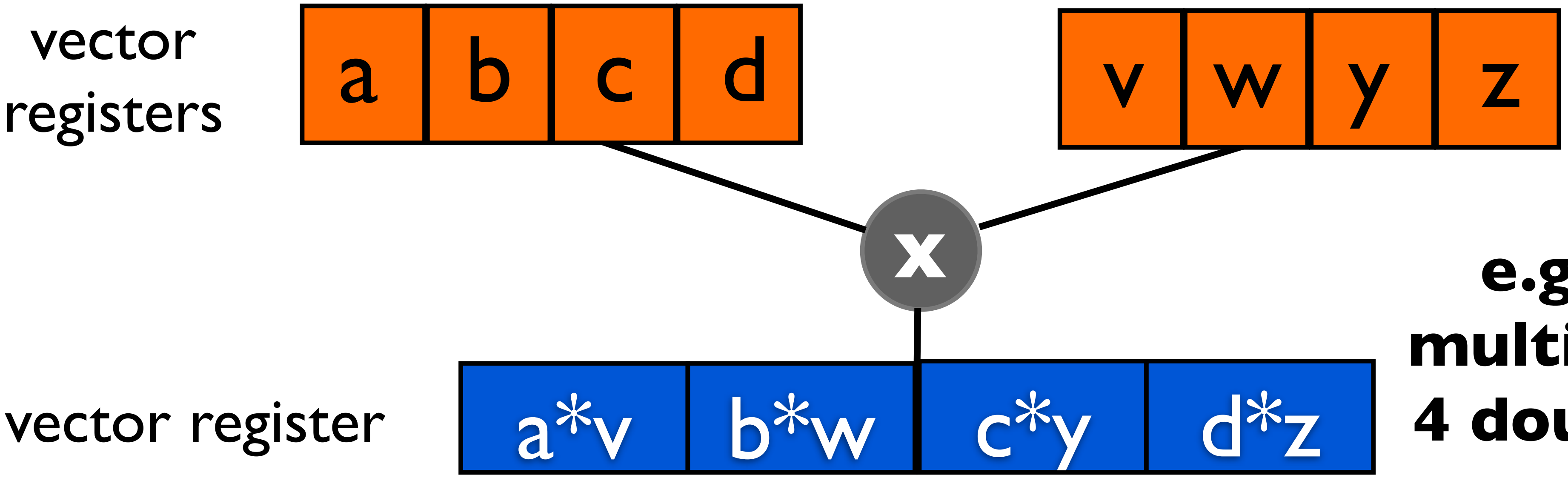
Reminder: Data Parallelism and SIMD instructions

- **why do we hope to gain from vector processing?**

Reminder: Data Parallelism and SIMD instructions

- **why do we hope to gain from vector processing?**

- “vectorized data” is a prerequisite to make efficient use of modern CPU vector instruction sets (“microparallelization”)

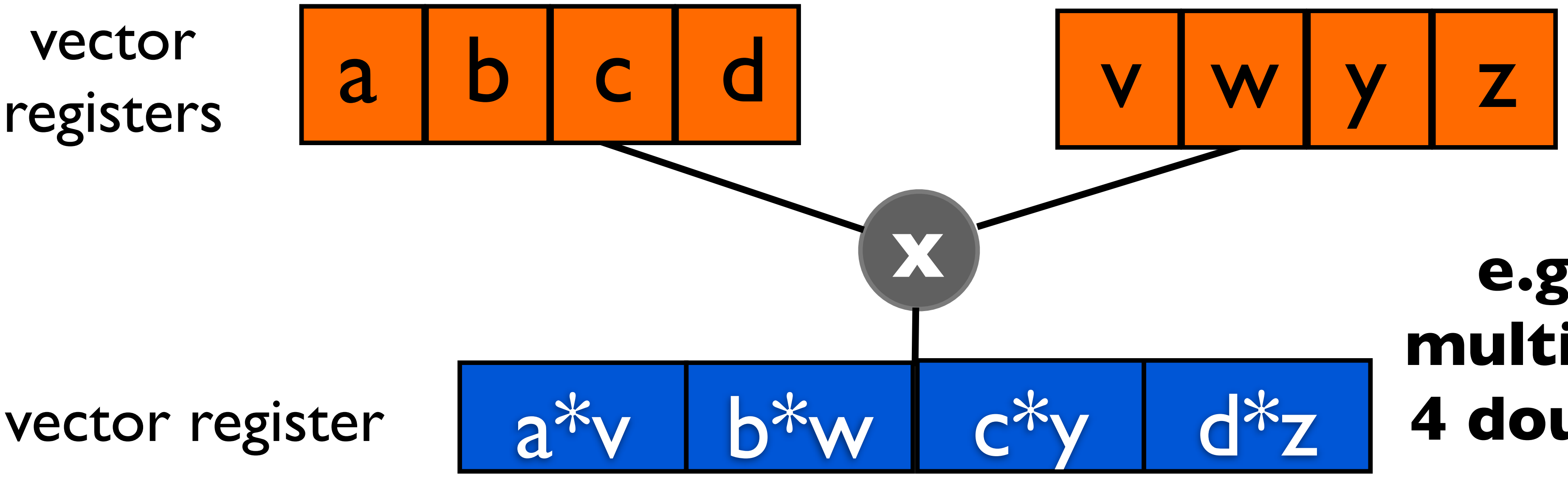


e.g., parallel multiplication of 4 doubles (AVX)

Reminder: Data Parallelism and SIMD instructions

- **why do we hope to gain from vector processing?**

- "vectorized data" is a prerequisite to make efficient use of modern CPU vector instruction sets ("microparallelization")



e.g., parallel multiplication of 4 doubles (AVX)

- effective use of vector instruction essential in order to make full use of current CPUs

- if we can gain from SIMD instructions, code is also **better prepared for accelerators** (GPUs, Intel Phi, ...)

Demonstrator Goals

Primary Goals

- setup simple vectorized geometry "demonstrator" to gain **development experience** and to **evaluate possible gains/pitfalls**
- focus for now on **CPU SIMD** opportunities (AVX instructions)
- vector optimize 3 geometries/shapes from simple to hard
  - **TGeoBBox**, TGeoCone, TGeoPolyCone
  - starting with TGeoBBox as most elementary block (needed also by more complicated shapes)

Demonstrator Goals

Primary Goals

- setup simple vectorized geometry "demonstrator" to gain **development experience** and to **evaluate possible gains/pitfalls**
- focus for now on **CPU SIMD** opportunities (AVX instructions)
- vector optimize 3 geometries/shapes from simple to hard
  - **TGeoBBox**, TGeoCone, TGeoPolyCone
  - starting with TGeoBBox as most elementary block (needed also by more complicated shapes)

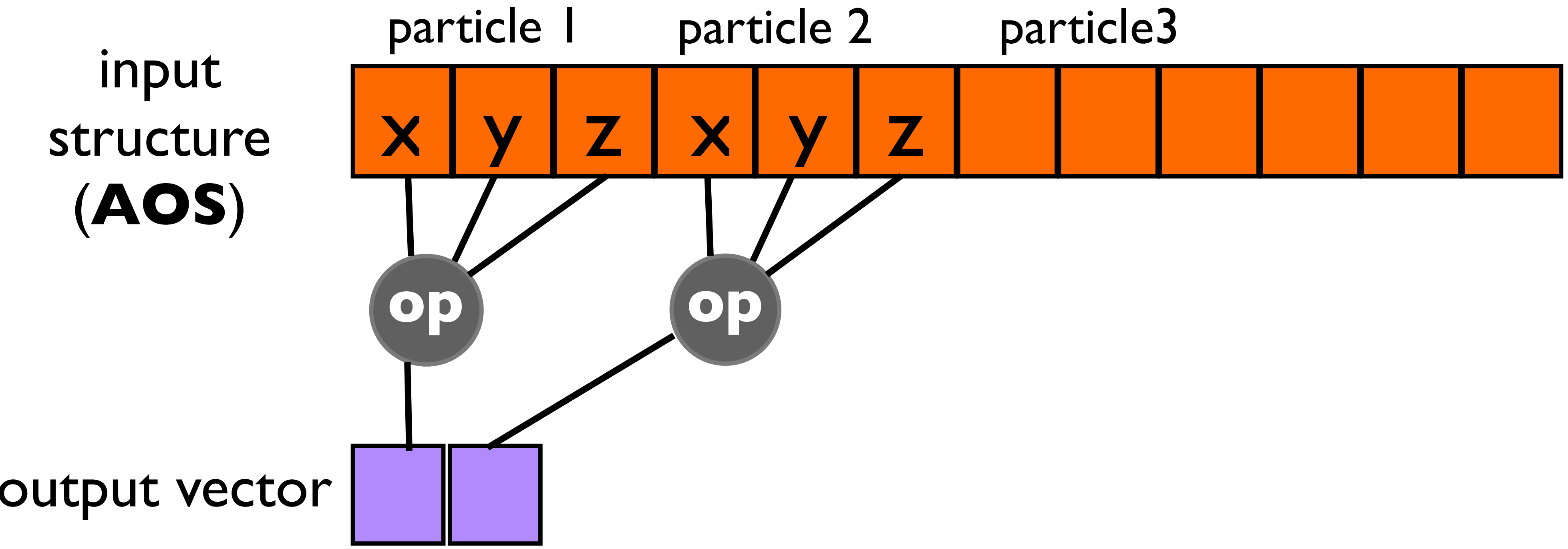
Requirements (wishes)

- rely on **existing code** base for geometry (here ROOT / USolid geometry)
- target compiler **autovectorization** whenever possible (in contrast to programming in intrinsics or some wrapper)
- compiler/platform **independence** ????

Some general things to fight with

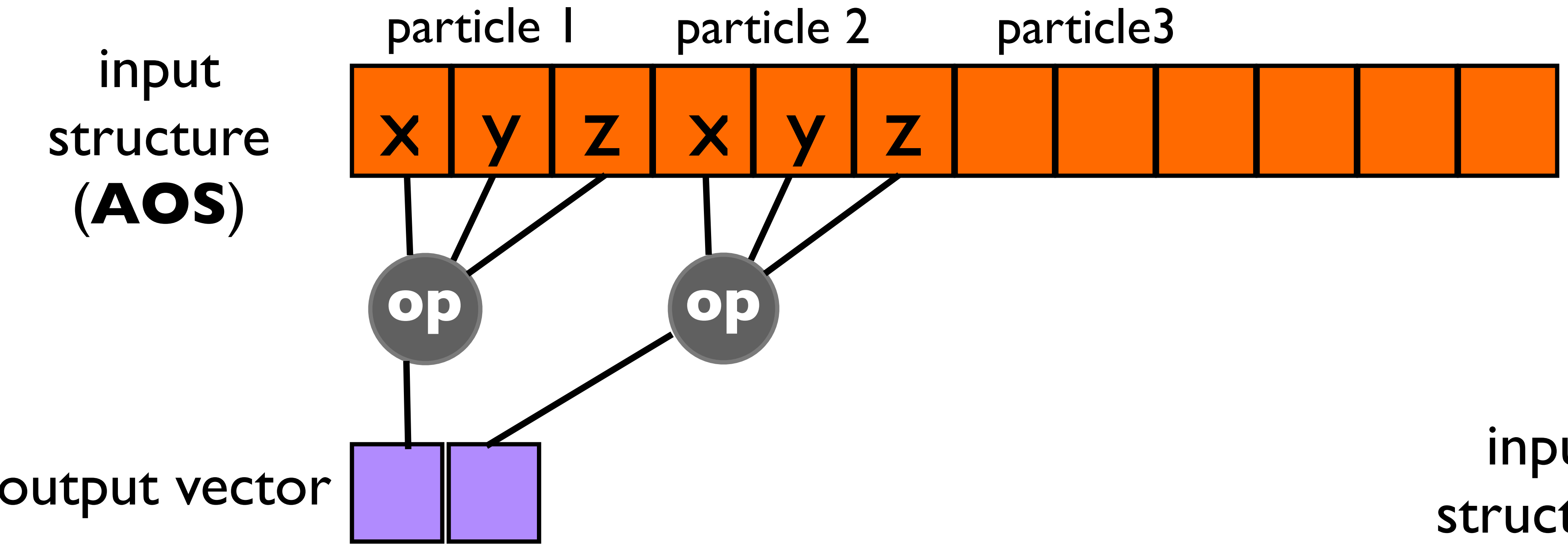
- refactoring code to make use of SIMD instructions can be a major undertaking ... (see next slides)
- unfortunately **considerable differences** between compilers ... bad when trying to develop generic code
- **Intel icc (v13)** generally better than **gcc/4.8**
  - but often only with the help of **“forcing” pragmas** (#pragma ivdep, #pragma simd)
- a particular difference seems to be vector element access patterns...

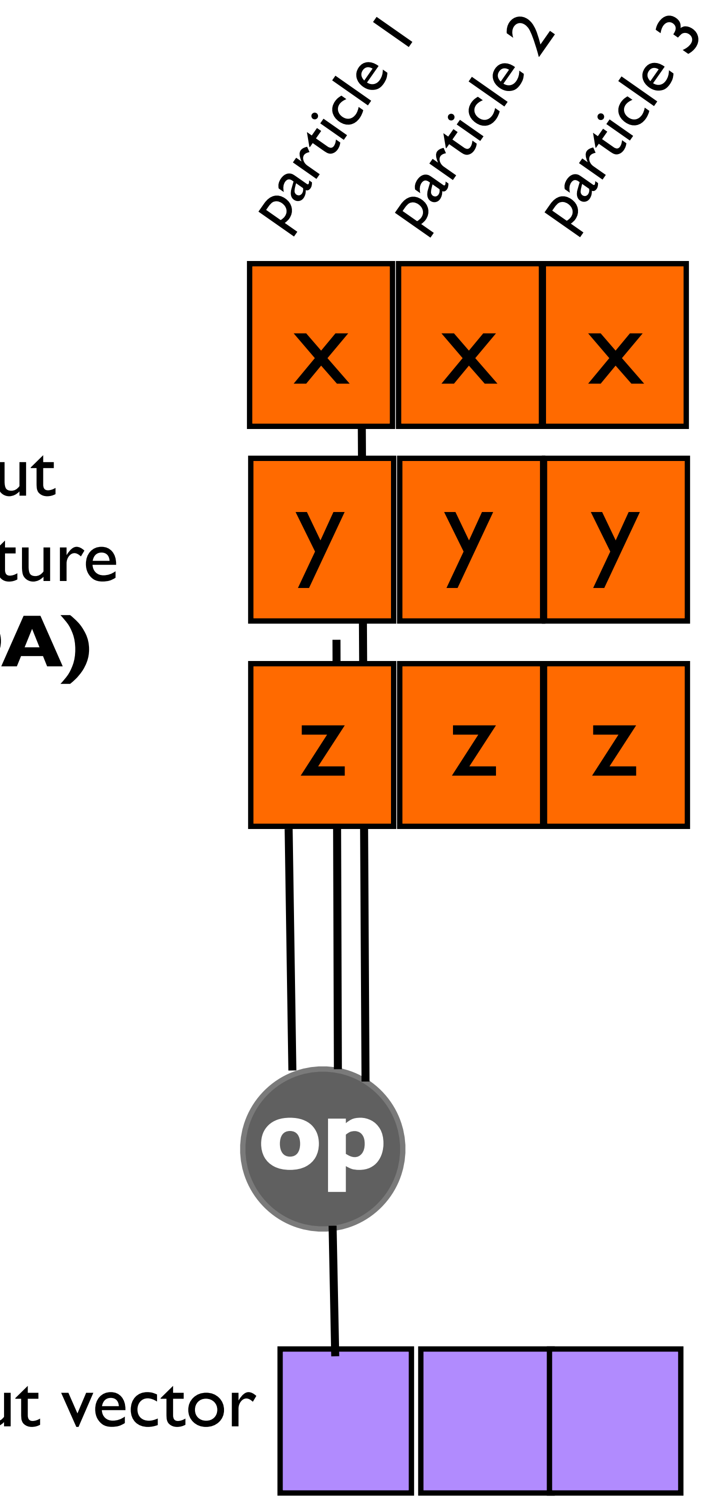
Memory Access Problem / Consideration



- a natural way to do vector processing of particles would be AOS approach
- above memory access pattern typical (3-to-1)
- **gcc/4.8 does not currently autovectorize** such patterns while **Intel icc can do it**

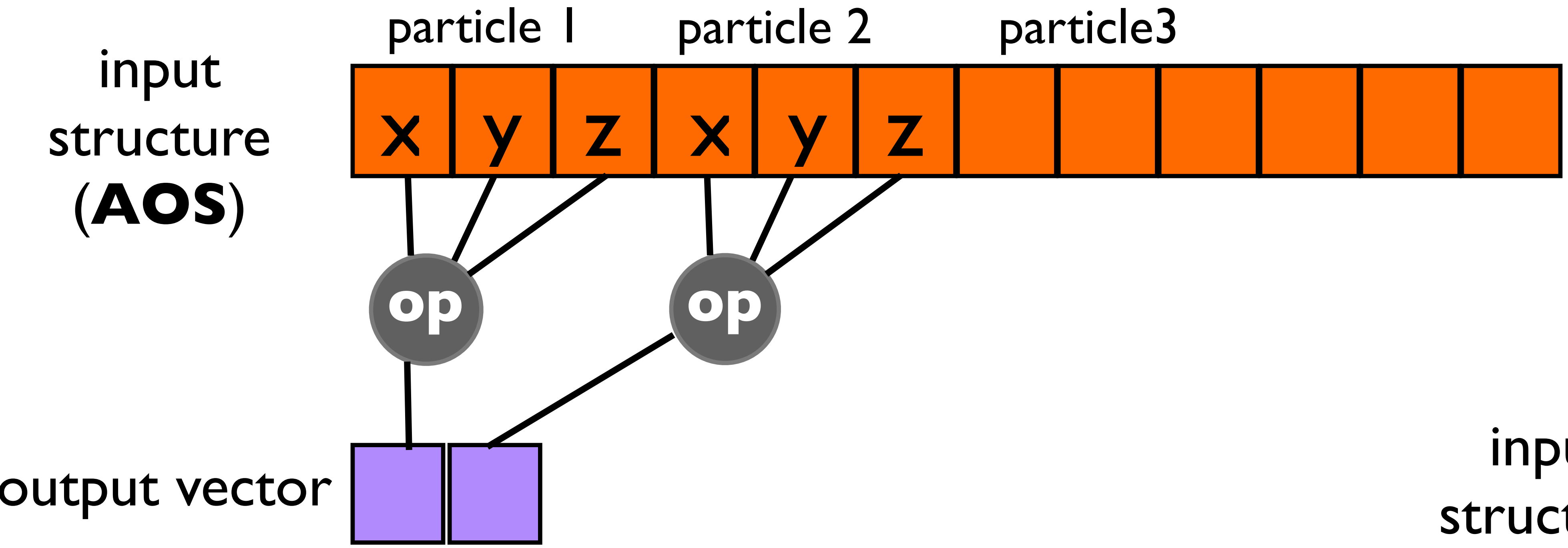
Memory Access Problem / Consideration

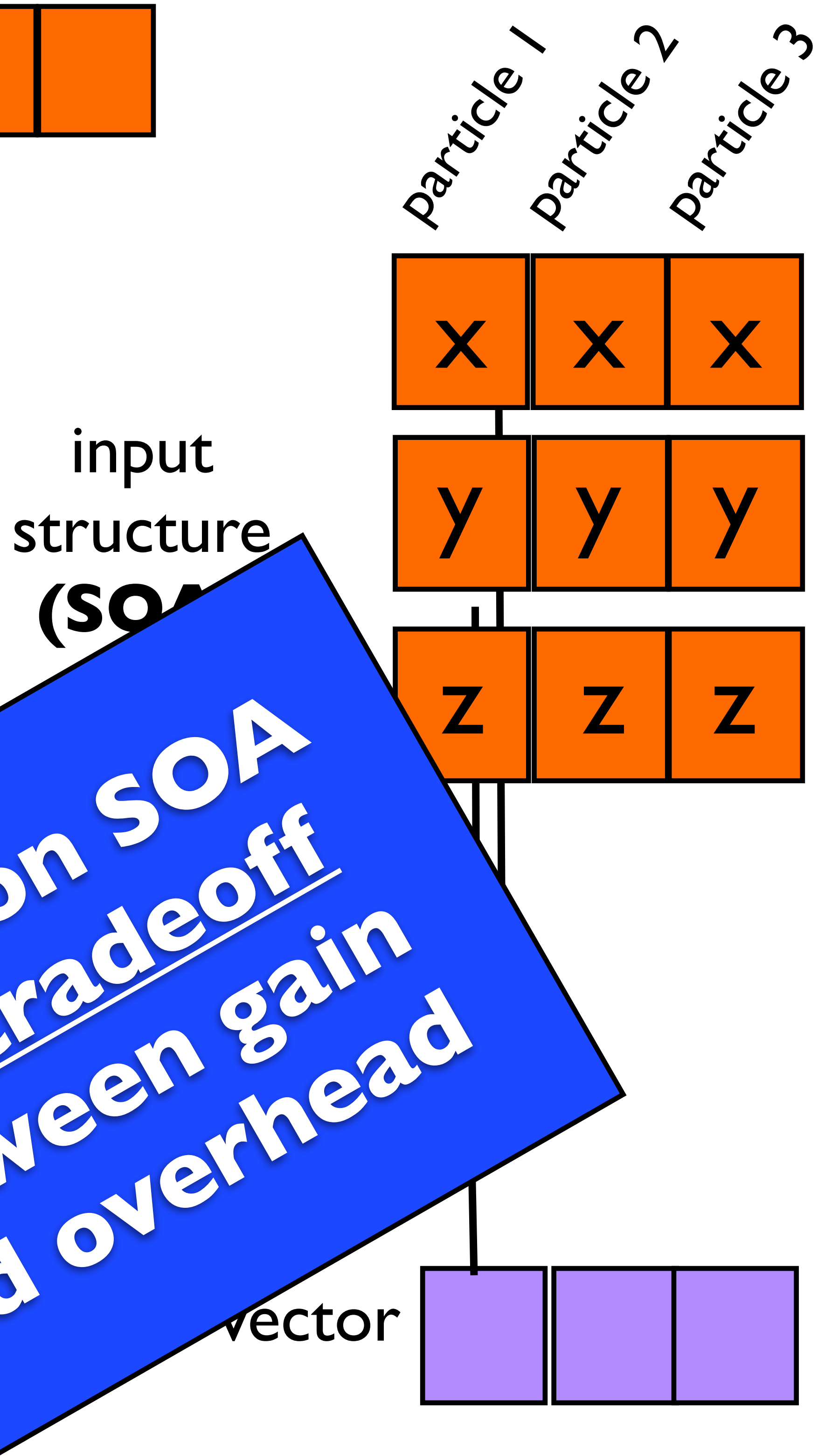




- a natural way to do vector processing of particles would be AOS approach
- above memory access pattern typical (3-to-1)
- **gcc/4.8 does not currently autovectorize** such patterns while **Intel icc can do it**
- SOA approach is better autovectorizable
- memory access in SOA pattern also more efficient

Memory Access Problem / Consideration





- a natural way to do vector processing of particles would be AOS approach

- above memory access pattern typical (?

- **gcc/4.8 does not currently aut** such patterns while **Intel icc can do**

- SOA approach is better autovectorizable

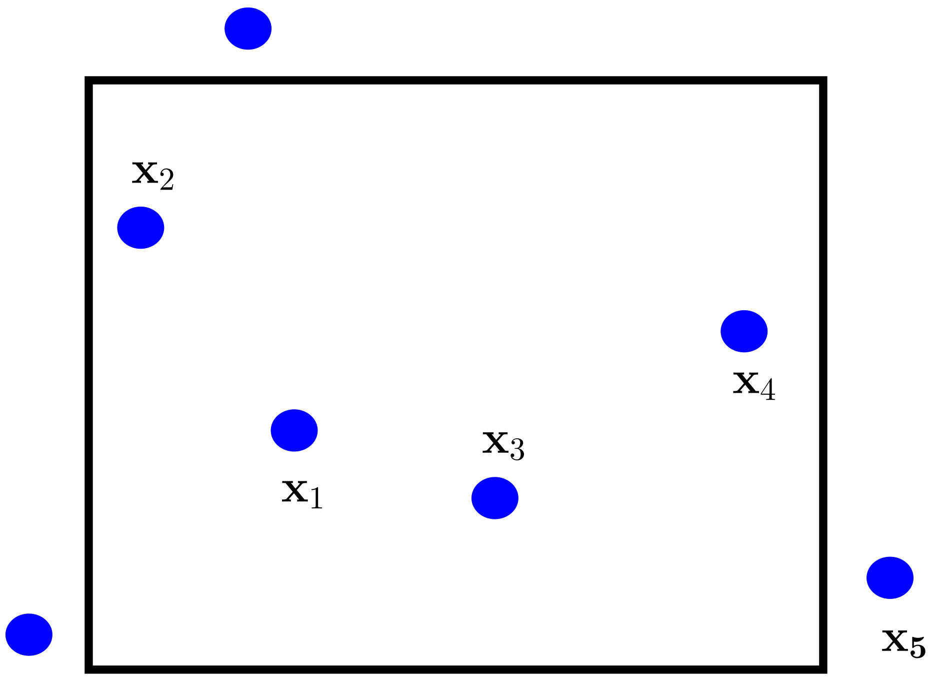
- memory access in SOA pattern also more effi nt

Focus on SOA but tradeoff between gain and overhead

Hurdles on the way to (auto)vectorization

- **starting point: some existing code** (here easy example)

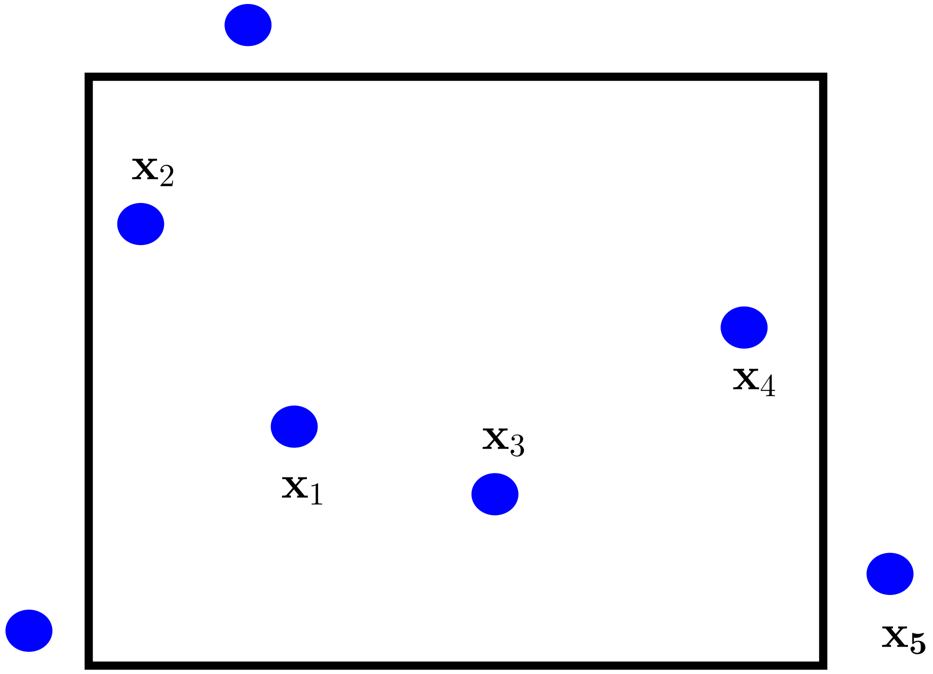
```
bool contains( const double * point ){
    for( unsigned int dir=0; dir < 3; ++dir ){
        if( fabs (point[dir]-origin[dir]) > boxsize[dir] )
        return false;
    }
    return true;
}
```



Hurdles on the way to (auto)vectorization

- **starting point: some existing code** (here easy example)

```
bool contains( const double * point ){
    for( unsigned int dir=0; dir < 3; ++dir ){
        if( fabs (point[dir]-origin[dir]) > boxsize[dir] )
        return false;
    }
    return true;
}
```



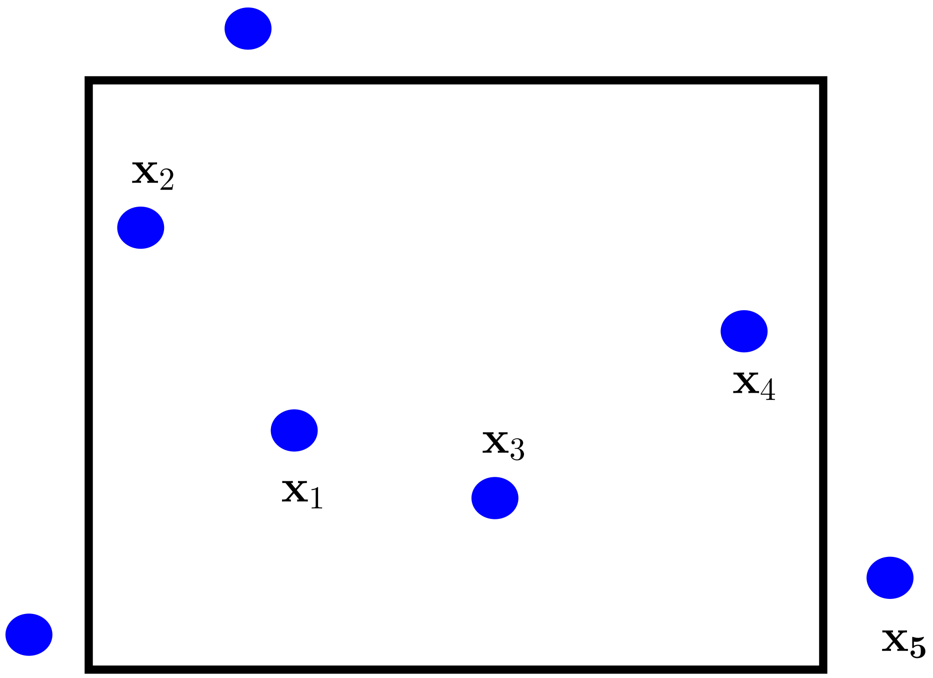
- **step 0: provide vector-interface**, call basic/elemental function ... and hope that compiler autovectorizes ...

```
void contains_v( const double * point, bool * isin, int np ) {
    for( unsigned int k=0; k < np; ++k) {
        isin[k]=contains( &point[3*k] );
}}
```

Hurdles on the way to (auto)vectorization

- **starting point: some existing code** (here easy example)

```
bool contains( const double * point ){
    for( unsigned int dir=0; dir < 3; ++dir ){
        if( fabs (point[dir]-origin[dir]) > boxsize[dir] )
        return false;
    }
    return true;
}
```



- **step 0: provide vector-interface**, call basic/elemental function ... and hope that compiler autovectorizes ...

```
void contains_v( const double * point, bool * isin, int np ) {
    for( unsigned int k=0; k < np; ++k) {
        isin[k]=contains( &point[3*k] );
}}
```

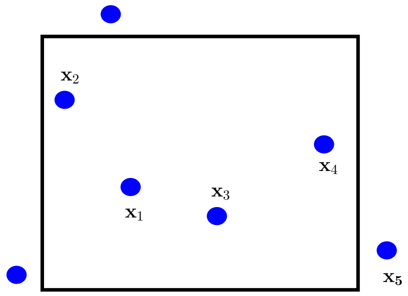
no auto-vectorization*

*icc: #pragma simd ok

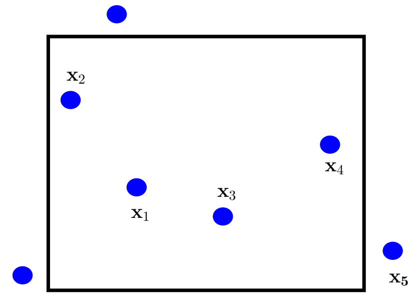
Hurdles on the way to (auto)vectorization (2)

- **step 1: inline and remove early returns**

```
void contains_v3( const double * point, bool * isin, int np ){
  for( unsigned int k=0; k < np; ++k){
     for( unsigned int dir=0; dir < 3; ++dir ){
     if ( fabs ( point[3*k+dir]-origin[dir] ) > boxsize[dir] ) isin[k]=false;
    }
    isin[k]=true;
}}
```



Hurdles on the way to (auto)vectorization (2)



step 1: inline and remove early returns

```
void contains_v3( const double * point, bool * isin, int np ){
  for( unsigned int k=0; k < np; ++k){
      for( unsigned int dir=0; dir < 3; ++dir ){
      if ( fabs ( point[3*k+dir]-origin[dir] ) > boxsize[dir] ) isin[k]=false;
   }
    isin[k]=true;
}}
```

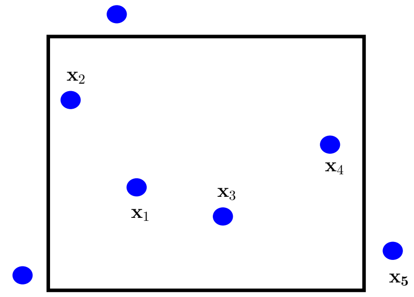
step 2: intermediate local variables + if conversion

```
 void contains_v4( const double * point, bool * isin, int np){
  for( unsigned int k=0; k < np; ++k){
     bool tmp[3]={true, true, true};
     for( unsigned int dir=0; dir < 3; ++dir ){
      tmp[dir] = fabs ( point[3*k+dir]-origin[dir] ) > boxsize[dir]; }
      isin[k]=tmp[0] & tmp[1] & tmp[2];
 }}
```

Hurdles on the way to (auto)vectorization (2)

step 1: inline and remove early returns

```
void contains_v3( const double * point, bool * isin, int np ){
  for( unsigned int k=0; k < np; ++k){
      for( unsigned int dir=0; dir < 3; ++dir ){
      if ( fabs ( point[3*k+dir]-origin[dir] ) > boxsize[dir] ) isin[k]=false;
   }
    isin[k]=true;
}}
```



step 2: intermediate local variables + if conversion

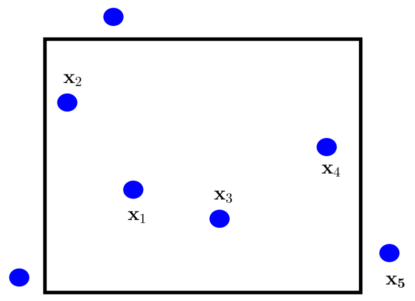
```
 void contains_v4( const double * point, bool * isin, int np){
  for( unsigned int k=0; k < np; ++k){
     bool tmp[3]={true, true, true};
     for( unsigned int dir=0; dir < 3; ++dir ){
      tmp[dir] = fabs ( point[3*k+dir]-origin[dir] ) > boxsize[dir]; }
      isin[k]=tmp[0] & tmp[1] & tmp[2];
 }}
```

no auto-vectorization*

*icc: #pragma ivdep ok

Hurdles on the way to (auto) vectorization (3)

- **step 3: AOS - SOA conversion**

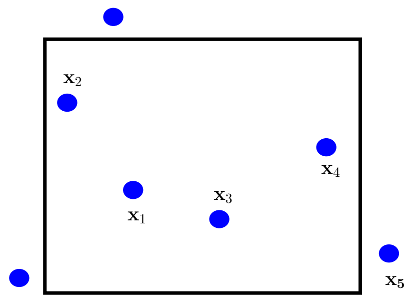


```
 typedef struct {
   double *coord[3];
 } P;

void contains_v6( const P &  point, bool * isin, int np ){
  for( unsigned int k=0; k < np; ++k){
      bool tmp[3];
      for( unsigned int dir=0; dir < 3; ++dir ){
        tmp[dir] = (fabs (point.coord[dir][k]-origin[dir]) > boxsize[dir]);
     }
      isin[k]=tmp[0] & tmp[1] & tmp[2];
}}
```

Hurdles on the way to (auto) vectorization (3)

- **step 3: AOS - SOA conversion**



```
typedef struct {
  double *coord[3];
} P;

void contains_v6( const P &  point, bool * isin, int np ){
 for( unsigned int k=0; k < np; ++k){
     bool tmp[3];
     for( unsigned int dir=0; dir < 3; ++dir ){
       tmp[dir] = (fabs (point.coord[dir][k]-origin[dir]) > boxsize[dir]);
    }
     isin[k]=tmp[0] & tmp[1] & tmp[2];
}}
```

no auto-vectorization*

*gcc/4.7 ok ?? , gcc/4.8 no

Hurdles on the way to (auto)vectorization (4)

- **step 4: (manually) unroll inner loops in source**

```
void contains_v7( const P & points, bool *  isin, int np ){
  for( unsigned int k=0; k < np; ++k)
  {
      bool resultx=(fabs (point.coord[0][k]-origin[0]) > boxsize[0]);
      bool resulty=(fabs (point.coord[1][k]-origin[1]) > boxsize[1]);
      bool resultz=(fabs (point.coord[2][k]-origin[2]) > boxsize[2]);
      isin[k]=resultx & resulty & resultz;
}}
```

Hurdles on the way to (auto)vectorization (4)

- **step 4: (manually) unroll inner loops in source**

```
void contains_v7( const P & points, bool *  isin, int np ){
  for( unsigned int k=0; k < np; ++k)
  {
      bool resultx=(fabs (point.coord[0][k]-origin[0]) > boxsize[0]);
      bool resulty=(fabs (point.coord[1][k]-origin[1]) > boxsize[1]);
      bool resultz=(fabs (point.coord[2][k]-origin[2]) > boxsize[2]);
      isin[k]=resultx & resulty & resultz;
}}
```

this auto-vectorizes !!

- **this is only version** that **autovectorizes uncondionally** with all compilers tested (icc 13, gcc 4.7/4.8)

- to arrive here, **needed ca. 4 refactoring steps**

many methods are MUCH!! more complicated

- (shortened) example from TGeoPCon:: DistFromOutside

```
double TGeoPcon::DistFromOutside(double *point, double *dir, double step){
    /// ....
    double r2 = point[0]*point[0]+point[1]*point[1];
    double radmax=0;
    radmax=fRmax[TMath::LocMax(fNz, fRmax)];
    if (r2>(radmax*radmax)) {
        double rpr=-point[0]*dir[0]-point[1]*dir[1];
        double nxy=dir[0]*dir[0]+dir[1]*dir[1];
        if (rpr<sqrt((r2-radmax*radmax)*nxy)) return TGeoShape::Big();
    }
    int ipl = TMath::BinarySearch(fNz, fZ, point[2]);
    int ifirst = ipl;
    if (ifirst<0) {
        ifirst=0;
    } else if (ifirst>=(fNz-1)) ifirst=fNz-2;
    // compute distance to boundary
    return DistToSegZ(point,dir,ifirst);
}
```

many methods are MUCH!! more complicated

- (shortened) example from TGeoPCon:: DistFromOutside

```
double TGeoPcon::DistFromOutside(double *point, double *dir, double step){
   /// ....
   double r2 = point[0]*point[0]+point[1]*point[1];
   double radmax=0;
   radmax=fRmax[TMath::LocMax(fNz, fRmax)];
   if (r2>(radmax*radmax)) {
      double rpr=-point[0]*dir[0]-point[1]*dir[1];
      double nxy=dir[0]*dir[0]+dir[1]*dir[1];
      if (rpr<sqrt((r2-radmax*radmax)*nxy)) return TGeoShape::Big();
   }
   int ipl = TMath::BinarySearch(fNz, fZ, point[2]);
   int ifirst = ipl;
   if (ifirst<0) {
      ifirst=0;
   } else if (ifirst>=(fNz-1)) ifirst=fNz-2;
   // compute distance to boundary
   return DistToSegZ(point,dir,ifirst);
}
```

1 **branches** / complicated **control flow**

many methods are MUCH!! more complicated

- (shortened) example from TGeoPCon:: DistFromOutside

```
double TGeoPcon::DistFromOutside(double *point, double *dir, double step){
   /// ....
   double r2 = point[0]*point[0]+point[1]*point[1];
   double radmax=0;
   radmax=fRmax[TMath::LocMax(fNz, fRmax)];
   if (r2>(radmax*radmax)) {
1     double rpr=-point[0]*dir[0]-point[1]*dir[1];
      double nxy=dir[0]*dir[0]+dir[1]*dir[1];
      if (rpr<sqrt((r2-radmax*radmax)*nxy)) return TGeoShape::Big();
   }
   int ipl = TMath::BinarySearch(fNz, fZ, point[2]);
   int ifirst = ipl;          2
   if (ifirst<0) {
1     ifirst=0;
   } else if (ifirst>=(fNz-1)) ifirst=fNz-2;
   // compute distance to boundary
   return DistToSegZ(point,dir,ifirst);
}             2
```

1 **branches** / complicated **control flow**

2 call to **other functions** (requires separate vectorization)

many methods are MUCH!! more complicated

- (shortened) example from TGeoPCon:: DistFromOutside

```
double TGeoPcon::DistFromOutside(double *point, double *dir, double step){
   /// ....
   double r2 = point[0]*point[0]+point[1]*point[1];
   double radmax=0;
   radmax=fRmax[TMath::LocMax(fNz, fRmax)];
   if (r2>(radmax*radmax)) {
      double rpr=-point[0]*dir[0]-point[1]*dir[1];
      double nxy=dir[0]*dir[0]+dir[1]*dir[1];
      if (rpr<sqrt((r2-radmax*radmax)*nxy)) return TGeoShape::Big();
   }
   int ipl = TMath::BinarySearch(fNz, fZ, point[2]);
   int ifirst = ipl;
   if (ifirst<0) {
      ifirst=0;
   } else if (ifirst>=(fNz-1)) ifirst=fNz-2;
   // compute distance to boundary
   return DistToSegZ(point,dir,ifirst);
}
```

1 **branches** / complicated **control flow**

2 call to **other functions** (requires separate vectorization)

3 call to **math functions** (requires vector math library, e.g.,VDT)

many methods are MUCH!! more complicated

- (shortened) example from TGeoPCon:: DistFromOutside

```
double TGeoPcon::DistFromOutside(double *point, double *dir, double step){
   /// ....
   double r2 = point[0]*point[0]+point[1]*point[1];
   double radmax=0;
   radmax=fRmax[TMath::LocMax(fNz, fRmax)];
   if (r2>(radmax*radmax)) {
      double rpr=-point[0]*dir[0]-point[1]*dir[1];
      double nxy=dir[0]*dir[0]+dir[1]*dir[1];
      if (rpr<sqrt((r2-radmax*radmax)*nxy)) return TGeoShape::Big();
   }
   int ipl = TMath::BinarySearch(fNz, fZ, point[2]);
   int ifirst = ipl;
   if (ifirst<0) {
      ifirst=0;
   } else if (ifirst>=(fNz-1)) ifirst=fNz-2;
   // compute distance to boundary
   return DistToSegZ(point,dir,ifirst);
}
```

1 **branches** / complicated **control flow**

2 call to **other functions** (requires separate vectorization)

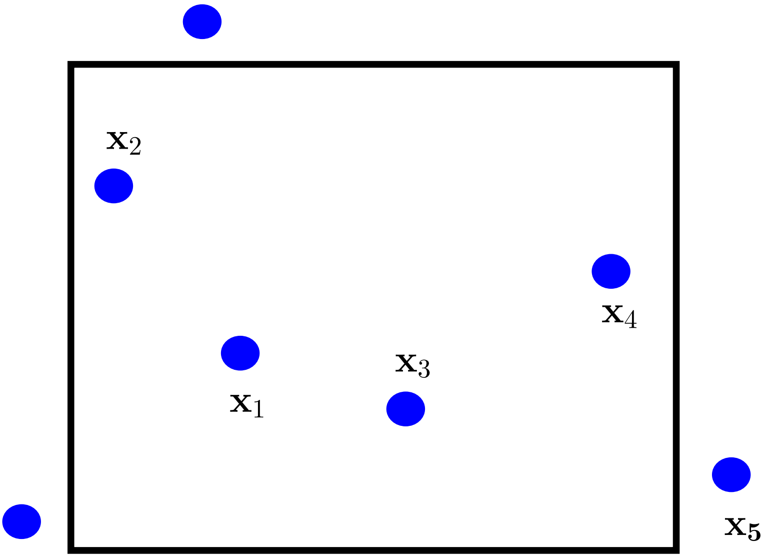
3 call to **math functions** (requires vector math library, e.g.,VDT)

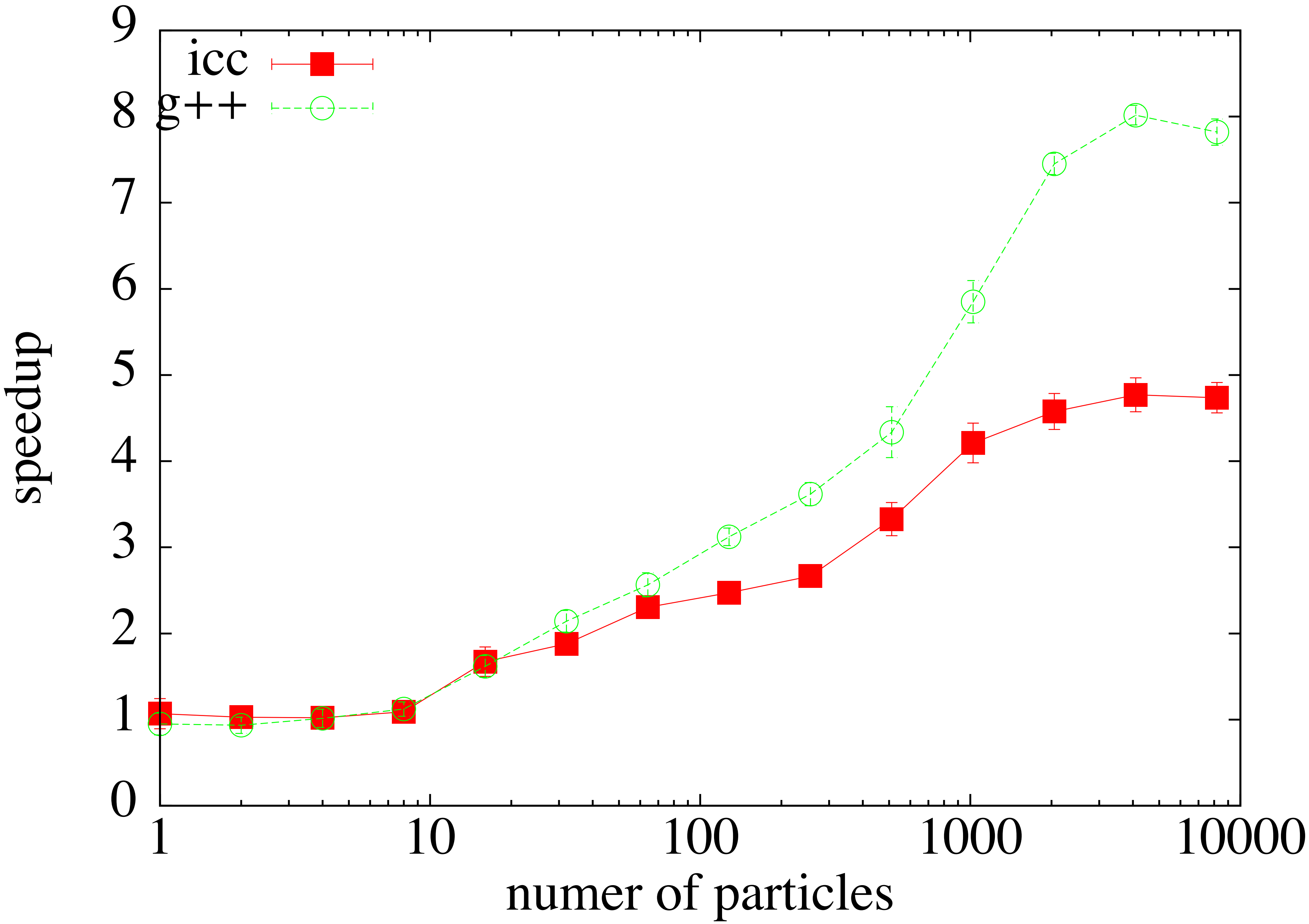
- we may be satisfied with **partial vectorization** (e.g., of expensive math functions)

First Performance Evaluation
-- Status for TGeoBBox --

TGeoBBox Performance Evaluation: Contains

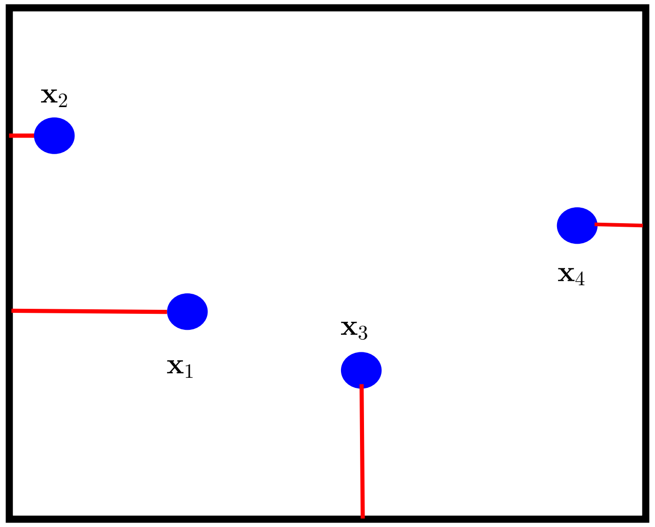
- calculates if particle inside box (see slides above)
- status: **both gcc/icc autovectorize, very good speedup**
- Intel iCore7 / AVX instructions

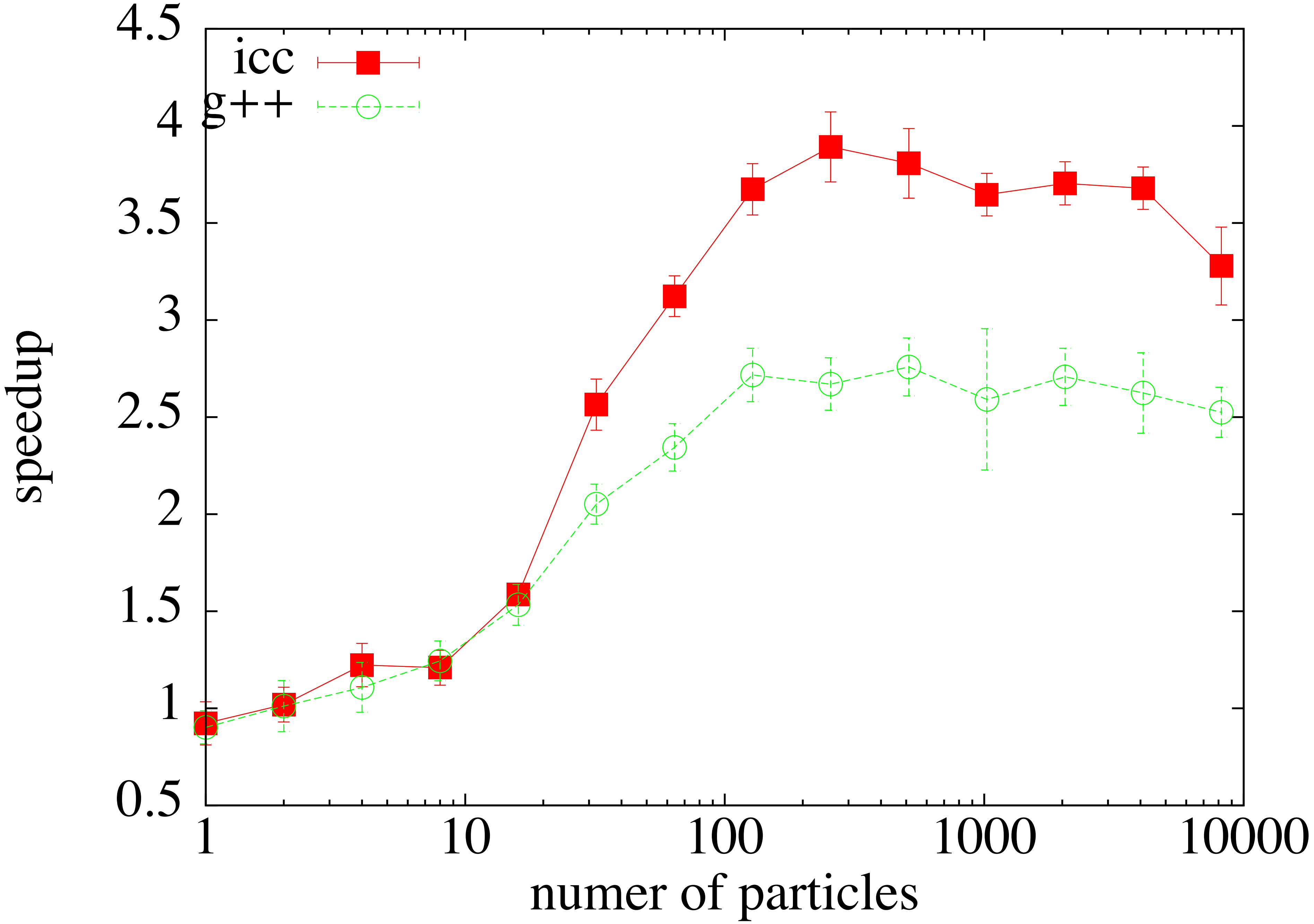




TGeoBBox Performance Evaluation: GetSafety

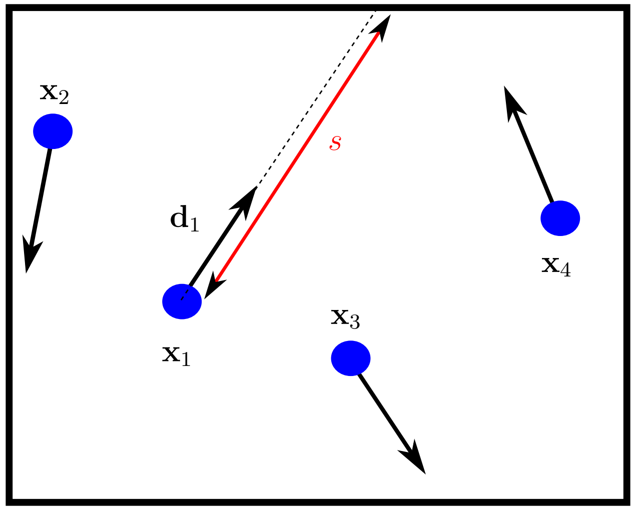
- calculates minimum distance to any surface
- status: **both gcc/icc autovectorize, good speedup**
- Intel iCore7 / AVX instructions

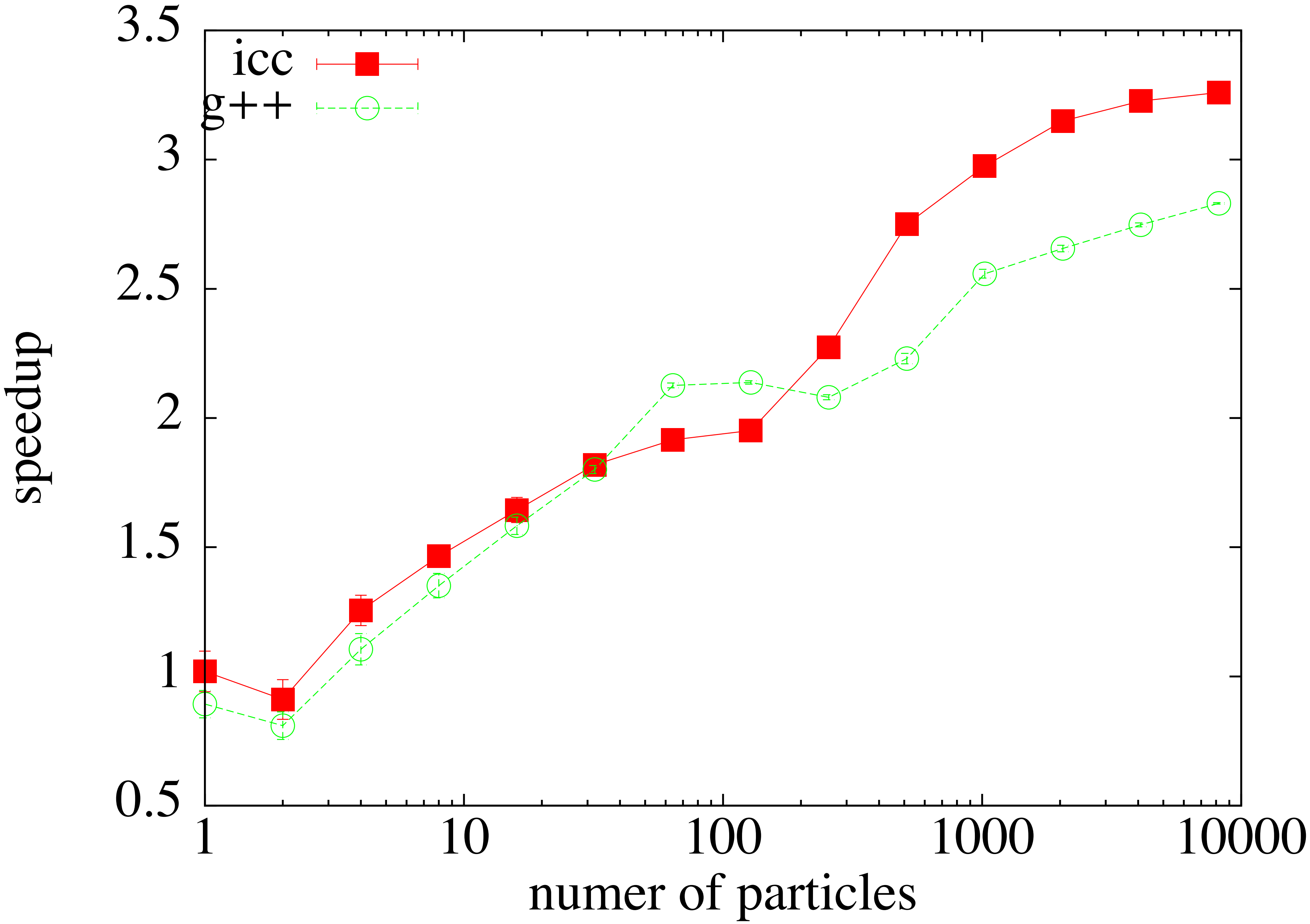




TGeoBBox Performance Evaluation: DistFromIn

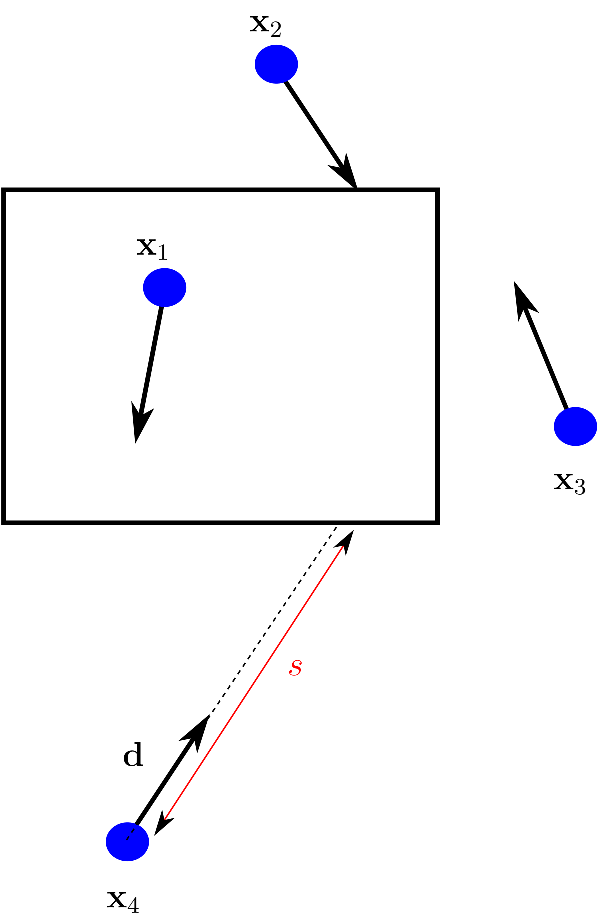
- calculates distance to hitting surface from inside
- status: **both gcc/icc autovectorize, good speedup**

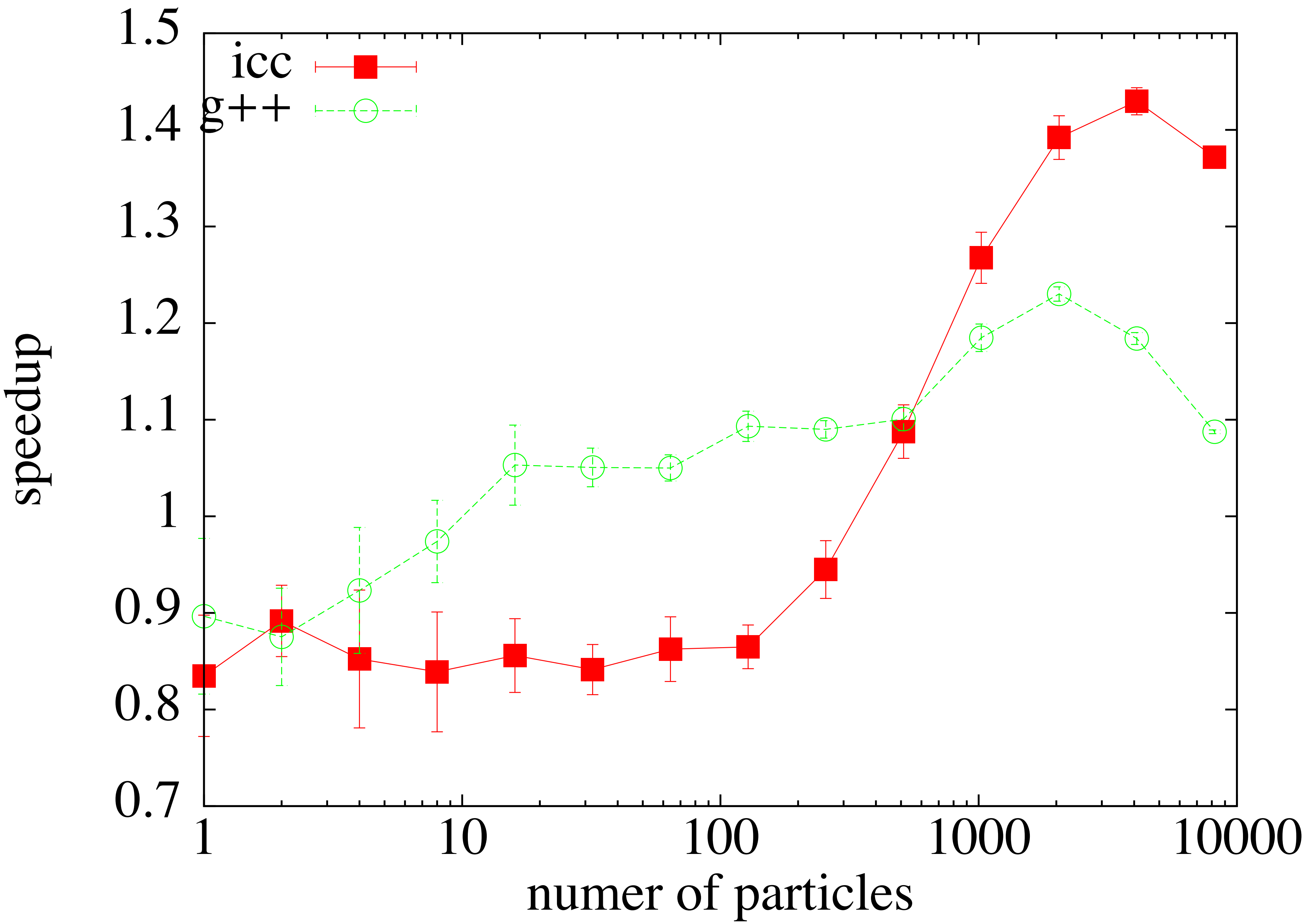




Performance Evaluation: DistFromOut

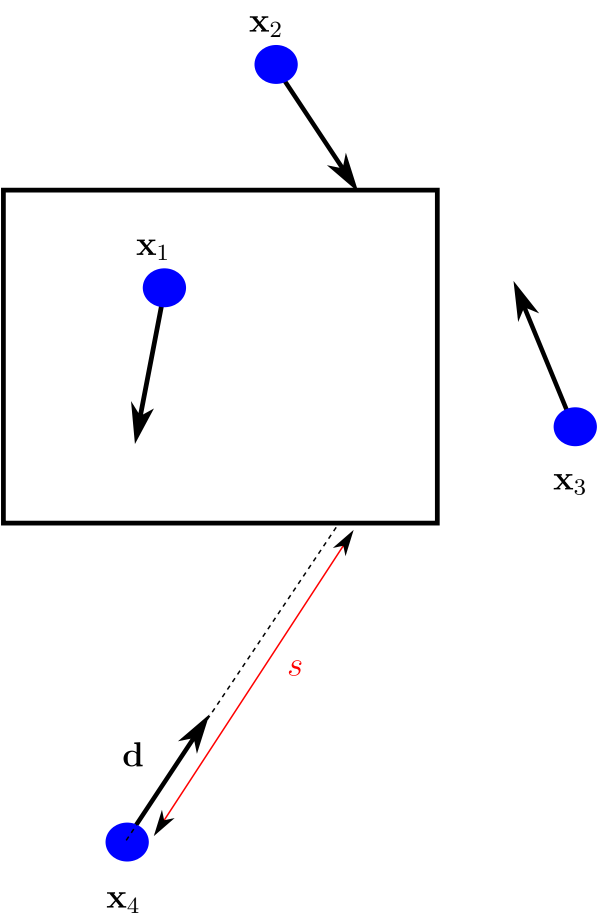
- calculates distance to hitting surface from outside
- status: **only icc autovectorizes** (too many conditionals??)
- initial performance **penalty** for small vectors (due to early return removals), some gain for larger number of particles

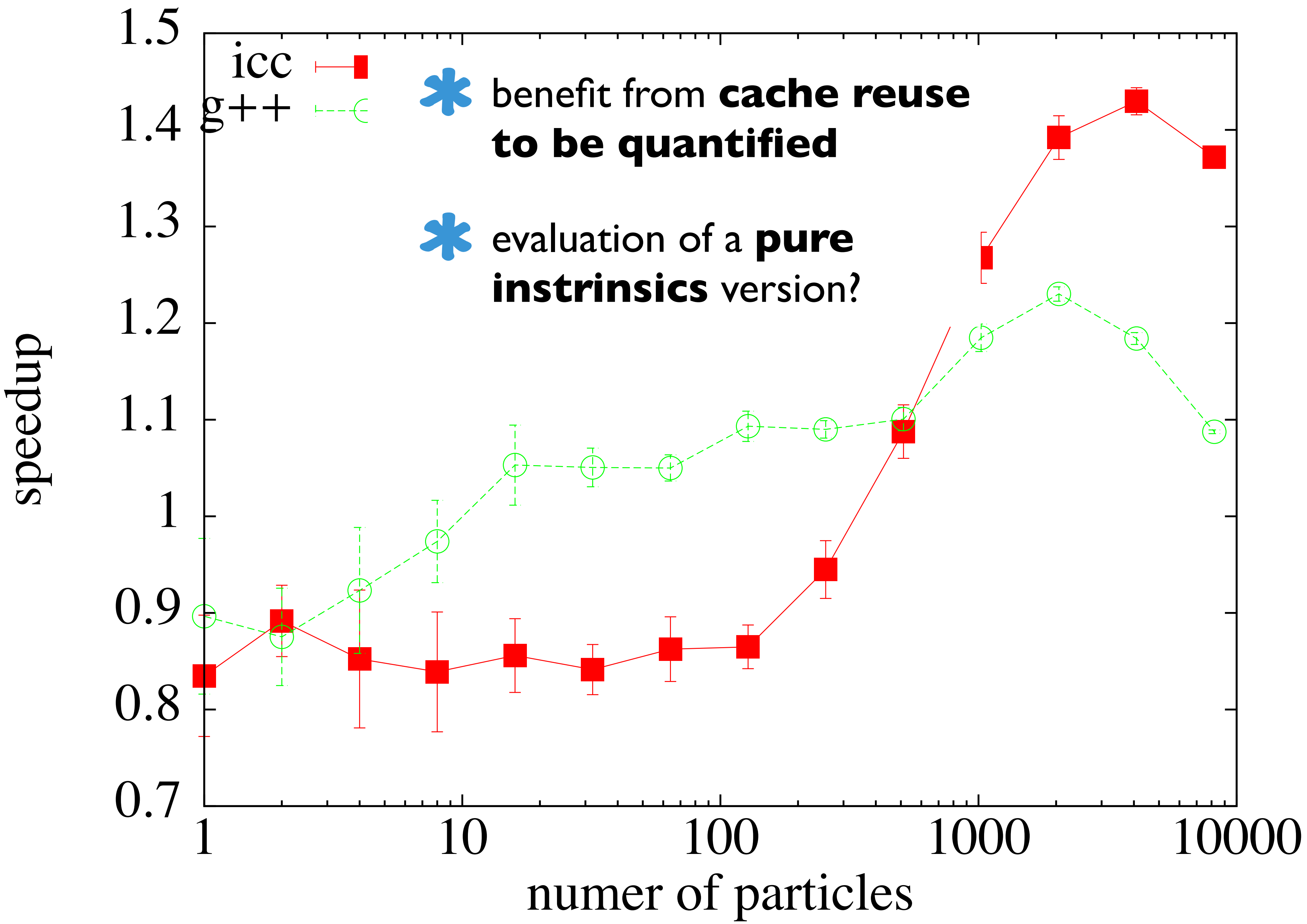




Performance Evaluation: DistFromOut

- calculates distance to hitting surface from outside
- status: **only icc autovectorizes** (too many conditionals??)
- initial performance **penalty** for small vectors (due to early return removals), some gain for larger number of particles





- benefit from **cache reuse to be quantified**
- evaluation of a **pure instrinsics** version?

Summary / Outlook

Summary

- talked about challenges in optimizing geometry routines for “vector particle processing”
- rather expensive work-process, complete code rewrite often unavoidable
- on average good SIMD performance results for TGeoBBox methods
- for more complex shapes to be seen ... hope to gain at least from better cache performance

Summary / Outlook

Summary

- talked about challenges in optimizing geometry routines for "vector particle processing"
- rather expensive work-process, complete code rewrite often unavoidable
- on average good SIMD performance results for TGeoBBox methods
- for more complex shapes to be seen ... hope to gain at least from better cache performance

Outlook

- more complex shapes
- GPUs, Xeon Phi
- evaluate **performance gains from cache reuse**
- quantification of SOA conversion overhead

Backup: A need for tools...

- converting code for data parallelism can be a pain ... (see challenges)
- would be nice to have better tool support for this task, helping at least with often recurring work

A possible direction:

- **source-to-source transformations** (preprocessing)
  - provide trivial vectorized code version of a function
  - unroll inner loops, rewrite early returns, ...
  - Clang/LLVM API very promising for this ... currently investigating

LLVM COMPILER INFRASTRUCTURE

- some tools go into this direction:
  - Scout (TU Dresden): Can take code within a loop and **emit instrinsics** code for all kinds of architectures
  - could be used in situations where the compiler does not autovectorize

SCOUT A Source-to-Source Transformator for SIMD-Optimizations

Backup: A need for tools...

- converting code for data parallelism can be a pain ... (see challenges)
- would be nice to have better tool support for this t at least with often recurring work

Stay tuned !!

A possible direction:

- **source-to-source transformations** (preprocessing)
  - provide trivial vectorized code version of a function
  - unroll inner loops, rewrite early returns, ...
  - Clang/LLVM API very promising for this ... currently investigating

LLVM COMPILER INFRASTRUCTURE

- some tools go into this direction:
  - Scout (TU Dresden): Can take code within a loop and **emit instrinsics** code for all kinds of architectures
  - could be used in situations where the compiler does not autovectorize

SCOUT A Source-to-Source Transformator for SIMD-Optimizations