CERN openlab Platform Competence Center

Vectorization with CilkPlus in Geant Vector Prototype

Juan José Fumero Alfonso

juan.jose.fumero.alfonso@cern.ch

CERN openlab summer student 2013

Outline

1. CilkPlus
2. Case of study: MxM and performance
3. Geant Vector Prototype CilkPlus approaches
4. Conclusions and contributions

CilkPlus Compiler

- It is an extension to the C/C++ languages to support data and tasks parallelism
- New tokens to express tasks parallelism
- New syntax to express data parallelism
- Quickest way to harness the power of both multicore and vector processing
- Intel C/C++ implementation and GCC-4.8.1 CilkPlus branch (last commits).

Case of Study: MxM OpenMP

```
void mxm_omp(double * restrict result, double *a, double *b, int m) {
    int i, j, k;
     #pragma omp parallel for private(i, j, k) firstprivate(m) \\
     shared(result,a,b)
    for (i = 0; i < m; i++) {
        for (j = 0; j < m; j++) {
            for (k = 0; k < m; k++) {
                result[i*m+j] += a[i*m+k] * b[k*m+j];
            }
        }
    }
}
```

MxM: CilkPlus Array Notation

```
void mxm_array_notation_fail(double  * restrict result, double * a,
double * b, int n) {
int stride = 4;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k+= stride) {
                result[i*n+j:stride] += a[i*n+k:stride] * b[k*n+j:stride:n];
            }
        }
    }
}
```

MxM: CilkPlus Array Notation

```
void mxm_array_notation_fail(double  * restrict result, double * a,
double * b, int n) {
int stride = 4;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k+= stride) {
                result[i*n+j:stride] += a[i*n+k:stride] * b[k*n+j:stride:n];
            }
        }
    }
}
```

1. **This code is not correct**
2. **“Result is invariant.**
3. **J columns of b would be processed at the same time. We need two strides at the same tima (k and j) => This is not possible with CilkPlus array notation.**
4. **The loop are unrolled for elements**
5. **We are multiplying wrong elements**
6. **A reduction operation is needed**

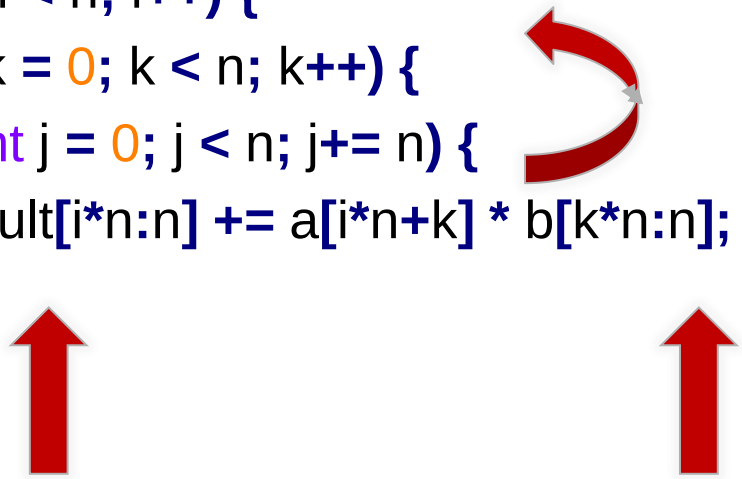
MxM Cilk+ - CilkArray

```
void mxm_cilkarray (double  * restrict result, double * a, double *
b, int n) {
int stride = 4;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k+= stride) {
   result[i*n+j] += __sec_reduce_add(a[i*n+k:stride] *
b[k*n+j:stride:n]);
            }
        }
    }
}
```

- **A gain is not expected because the only way to use array notation is in the reduction**
- **Also, the memory access pattern is not very optimal**
- **In this case we need to look for proper value of stride**

MxM Cilk+ Interchange

```
void mxm_array_notation_interchange(double  * restrict result, double *
a, double * b, int n) {
for (int i = 0; i < n; i++) {
        for (int k = 0; k < n; k++) {
            for (int j = 0; j < n; j+= n) {
                result[i*n:n] += a[i*n+k] * b[k*n:n];
            }
        }
    }
}
```



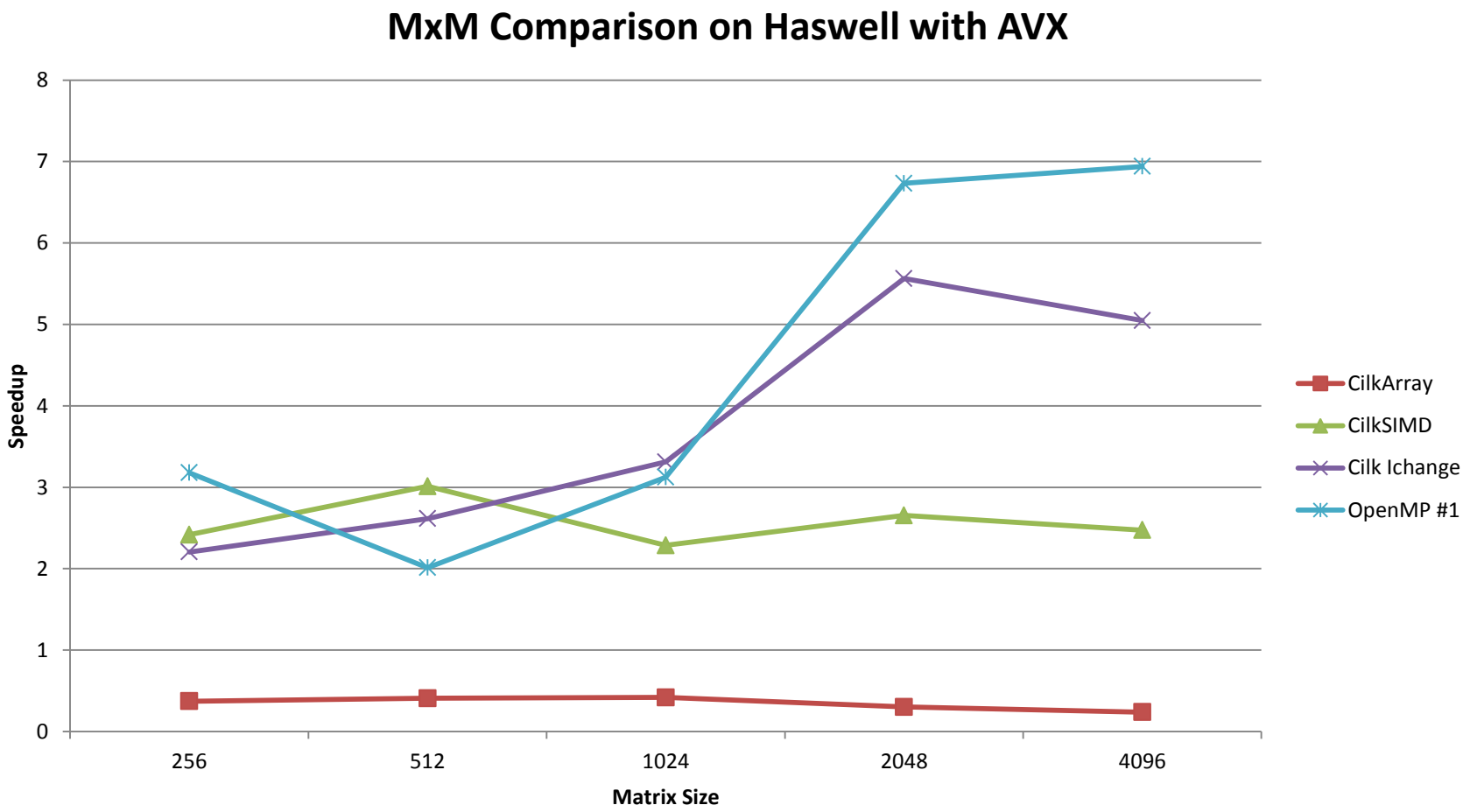
- **The stride now is not needed**
- **Result and B arrays are expressed with array notation**
- **Memory access patter more efficient**

… and #pragma simd ?

```
void mxm_cilk_simd(double  * restrict result, double * a, double * b, int n) {
for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         #pragma vector aligned
         #pragma simd reduction(+:result[i*n+j])
         for (int k = 0; k < n; k ++) {
            result[i*n+j] += a[i*n+k] * b[k*n+j];
         }
      }
   }
}
```

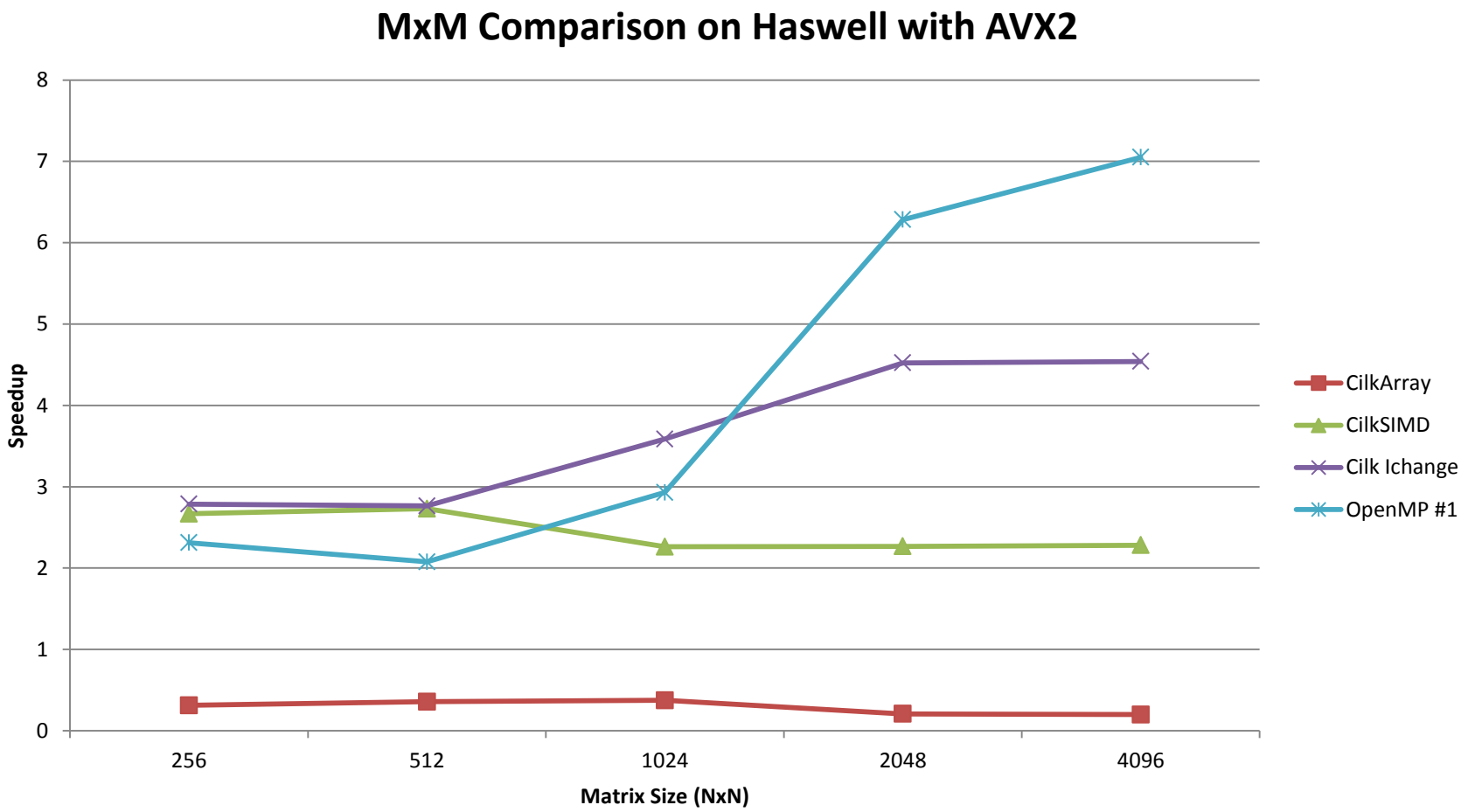
- **Reduction operation required in ICC 14.0 beta**
- **Vector aligment required => all vector computation instruction together.**

Speedup Cilk+ MxM,AVX Haswell



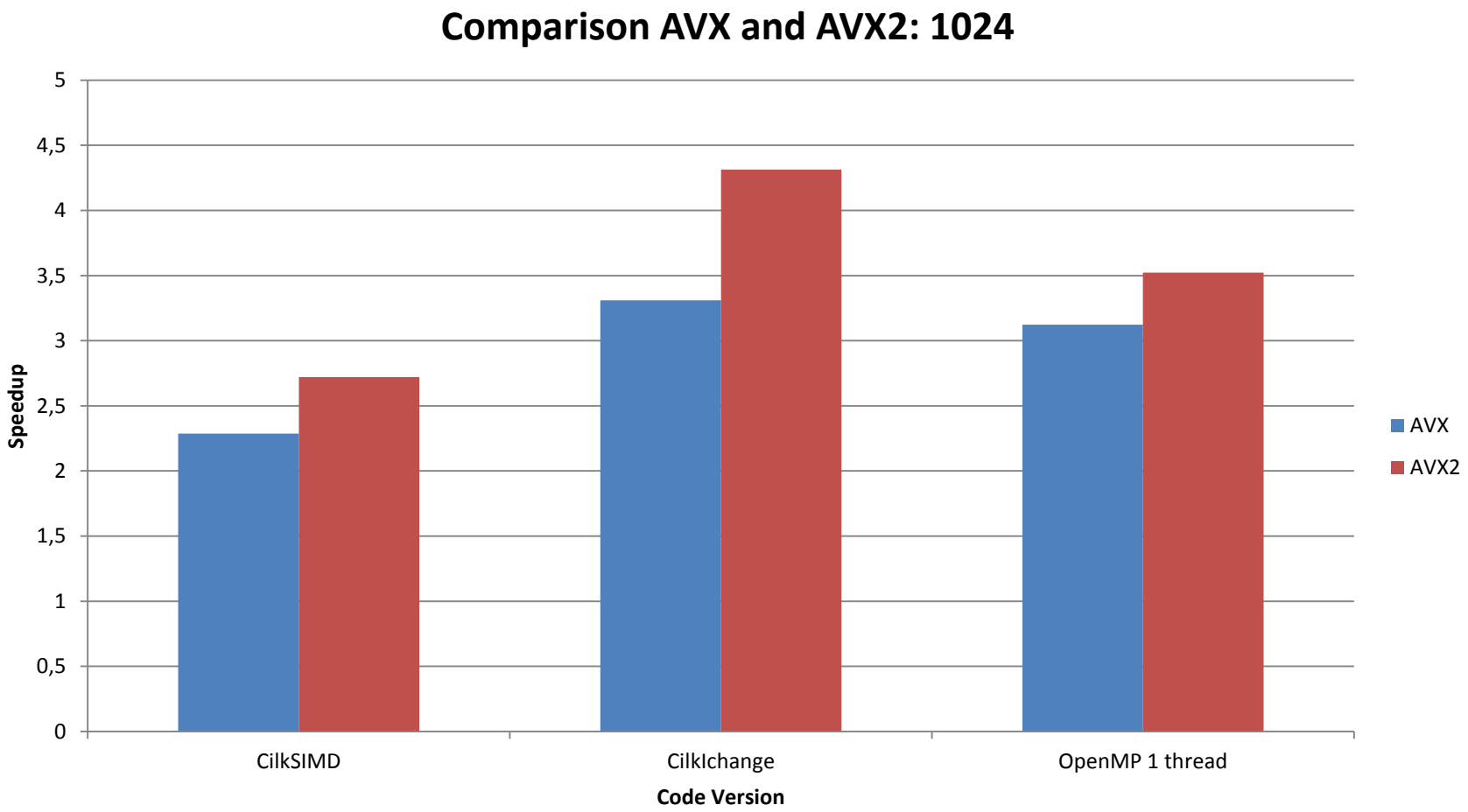
Intel(R) Xeon(R) CPU E3-1285L v3 @ 3.10GHz

Speedup Cilk+ MxM,AVX2 Haswell



Intel(R) Xeon(R) CPU E3-1285L v3 @ 3.10GHz

CilkPlus Comparison on Haswell



Intel(R) Xeon(R) CPU E3-1285L v3 @ 3.10GHz

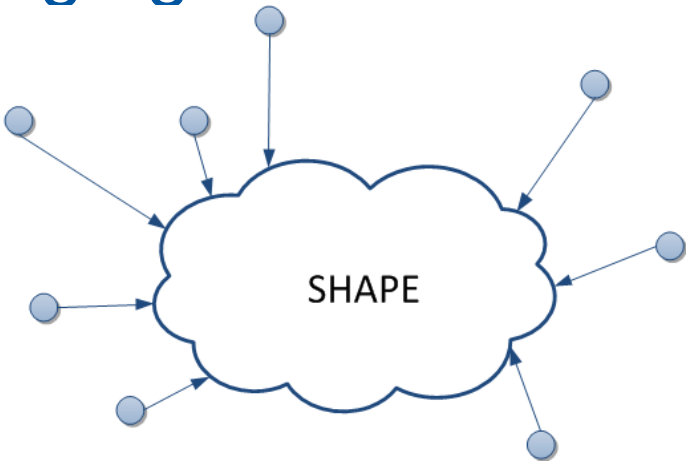
If (SIMD == Array Notation)

- ***#pragma simd*** instructs the compiler to ignore every implication from the language standard, such **as data dependencies, orders of memory access, etc**. It also turns off the compiler's efficiency heuristic.

- The array notation version itself still respects the language standards such as aliasing, data dependencies.

Geant Vector Prototype

- Detector simulation is one of the most CPU
- Intensive tasks in modern High Energy Physics.
- Geant5 represents a benchmark prototype aims leveraging vector instructions



Distance From Outside of a Shape

- Original version: ROOT Framework
- First Vectorizable Version: which can be used as a test to study CilkPlus performance (*)
- Five different implementations
  1. CilkPlus1: using small vectors
  2. CilkPlus2: array notation processing over all particles
  3. CilkPlus3: tiling over all particles
  4. CilkPlus4: Mallocs outside the main loop and tiling.
  5. CilkPlus5: smalls vectors with 4 particles each iteration.

(*) First Vectoriable Version from Sandro Wenzel.

Original code (sketch)

```
newpt[0] = p[0][k] - origin[0];
saf[0] = TMath::Abs(newpt[0])-par[0];
factor = (saf[0]>=stepmax[k]) ? TGeoShape::Big() : 1
 in = (saf[0]<0);

newpt[1] = p[1][k] - origin[1];
saf[1] = TMath::Abs(newpt[1])-par[1];
factor *= (saf[1]>=stepmax[k]) ? TGeoShape::Big() : 1;
in = in & (saf[1]<0);

newpt[2] = p[2][k] - origin[2];
saf[2] = TMath::Abs(newpt[2])-par[2];
factor *= (saf[2]>=stepmax[k]) ? TGeoShape::Big() : 1.;
in = in & (saf[2]<0);
```

DistanceFromOutside (Cilk1)

```
newpt[0] = p[0][k] - origin[0];
saf[0] = TMath::Abs(newpt[0])-par[0];
factor = (saf[0]>=stepmax[k]) ?
TGeoShape::Big() : 1
 in = (saf[0]<0);

newpt[1] = p[1][k] - origin[1];
saf[1] = TMath::Abs(newpt[1])-par[1];
factor *= (saf[1]>=stepmax[k]) ?
TGeoShape::Big() : 1;
in = in & (saf[1]<0);

newpt[2] = p[2][k] - origin[2];
saf[2] = TMath::Abs(newpt[2])-par[2];
factor *= (saf[2]>=stepmax[k]) ?
TGeoShape::Big() : 1.;
in = in & (saf[2]<0);
```

First Vectorizable Version (autovectorization)

```
factor[:] = 1;
double point_aux[3]    = { p[0][k], p[1][k], p[2][k]
};

#pragma vector align
newpt[0:3] = point_aux[0:3] - origin[0:3];
saf[:] = abs(newpt[:])-par[:];
if ( saf[:] >= stepmax[k] )
     factor[:] = geobig;
 else
     factor[:] = 1;
factor_scalar = __sec_reduce_mul(factor[:]);
```

Cilk1: small vectors

DistanceFromOutside (Cilk2)

```
factor[:] = 1;
double point_aux[3]    = { p[0][k],
p[1][k], p[2][k] };

#pragma vector align
newpt[0:3] = point_aux[0:3] -
origin[0:3];
saf[:] = abs(newpt[:])-par[:];
if ( saf[:] >= stepmax[k] )
      factor[:] = geobig;
 else
      factor[:] = 1;
factor_scalar =
__sec_reduce_mul(factor[:]);
```

Cilk1 (small vectors)

```
Double_t *newpt_x = (Double_t *)
  mm_malloc(sizeof(Double_t)*np,
ALIGN_AVX);

newpt_x[0:np] = p[0][0:np] - origin[0];
saf_vector_x[0:np] =
abs(newpt_x[0:np]) - par[0];

if (saf_vector_x[0:np]>=stepmax[0:np])
        factor[0:np] = big;
 else
       factor[0:np] = 1;
  if (saf_vector_x[0:np] < 0)
       in_vector[0:np] = true;
  else
       in_vector[0:np] = false;
```

Cilk2: processing all particles

Thinking …

DistanceFromOutside (CilkLast)

```
factor[:] = 1;
double point_aux[3]    = { p[0][k], p[1][k],
p[2][k] };

#pragma vector align
newpt[0:3] = point_aux[0:3] - origin[0:3];
saf[:] = abs(newpt[:])-par[:];
if ( saf[:] >= stepmax[k] )
     factor[:] = geobig;
 else
     factor[:] = 1;
factor_scalar =
__sec_reduce_mul(factor[:]);
```

Cilk1: small vectors

```
for(unsigned int k = 0; k < np; k += 4) {
      factor[:] = 1;
      #pragma simd
      for (unsigned int i = 0; i <
(LENVEC*3); i+=3) {
          point[i] = p[0][j];
          point[i+1] = p[1][j];
          point[i+2] = p[2][j];
         …
      j++;
      }
     #pragma vector align
     newpt[:] = point[:] - origin[:];
     saf[:] = abs(newpt[:])-par[:];
      for (unsigned int i= 0; i < 4; i++)
scalar[i] =  __sec_reduce_mul(factor[i*3:3]);
```

CilkPlus5: small vectors (4 elements each iteration)

Preliminary results

Version	Total time	Time (s) per particle	Same result?	Vector CODE	Vector ALL
ROOT	0,276	2,730E-04	YES	-	-
First Vector Version in Geant Vector Prototype	0,123	1,180E-04	YES	AVX	NO
CilkPlus 5	0,269	2,680E-04	YES	AVX	YES
CilkPlus 1	0,274	2,730E-04	YES	AVX	NO
CilkPlus 2	0,377	3,540E-04	YES	AVX	NO
CilkPlus 3	0,407	3,730E-04	YES	AVX	NO
CilkPlus 4	0,650	8,040E-04	NO	AVX	NO

Intel(R) Xeon(R) CPU E3-1285L v3 @ 3.10GHz

Perf analysis (I)

Version	Cycles	Insns Per Cycle	Branches	Time
First Vector Version in Geant Vector Prototype	2.678.846.562	1,17	481.479.014	0,000118
CilkPlus 5	3.340.918.582	1,12	545.952.203	0,000268
CilkPlus 1	3.356.677.160	1,24	640.323.963	0,000273
CilkPlus 2	3.870.739.549	1,07	634.312.969	0,000354
CilkPlus 3	3.950.685.971	1,08	651.628.305	0,000373
CilkPlus 4	4.715.796.328	1,19	645.794.406	0,000804

Intel(R) Xeon(R) CPU E3-1285L v3 @ 3.10GHz

Perf analysis (II)

Version	Cache Misses	Cache references	% Cache Misses	Time
First Vector Version in Geant Vector Prototype	327.094	4.509.255	7,254	0,000118
CilkPlus 5	894.158	6.879.908	12,997	0,000268
CilkPlus 1	887.427	6.699.531	13,246	0,000273
CilkPlus 2	3.169.916	13.064.301	24,264	0,000354
CilkPlus 3	3.396.769	13.423.264	25,305	0,000373
CilkPlus 4	3.057.942	14.076.222	21,724	0,000804

Intel(R) Xeon(R) CPU E3-1285L v3 @ 3.10GHz

Looking the ASM code

Code	#Lines	XMM Instructions	YMM instructions	V*tf128	Jump	Time
First Vector Version Geant Vector Pro.	729	434	135	33	28	0,150
CilkPlus 5	507	342	154	34	10	0,281
CilkPlus 1	289	133	0	0	11	0,266
CilkPlus 2	707	275	136	28	12	0,431
CilkPlus 3	707	275	136	28	12	0,462
CilkPlus 4	1010	406	217	102	44	0,745

Intel(R) Xeon(R) CPU E3-1285L v3 @ 3.10GHz

Conclusions and future

- The vector intruction set AVX and AVX2 show very good performance in combination with FMA for MxM problem
- CilkPlus is easy to use but to get good performance it is needed to know low level details
- Pragma simd is not exactly the same that array notation
- The CilkPlus compiler is very conservative to introduce AVX instructions set
- Exploring Intel Xeon PHI?

Contributions

- We have discovered some bugs in GCC CilkPlus branch. Now is fixed. (svn commits: r201772 - r201776)
- We have discovered some bugs in Intel CilkPlus implementation related with the parallel programming model. (Intel 14.0 beta)
- Report for Intel
- Better understanding with CilkPlus (Intel Cilkplus Team and GCC CilkPlus team)

Comments, suggestions, …

Thank you

Juan José Fumero

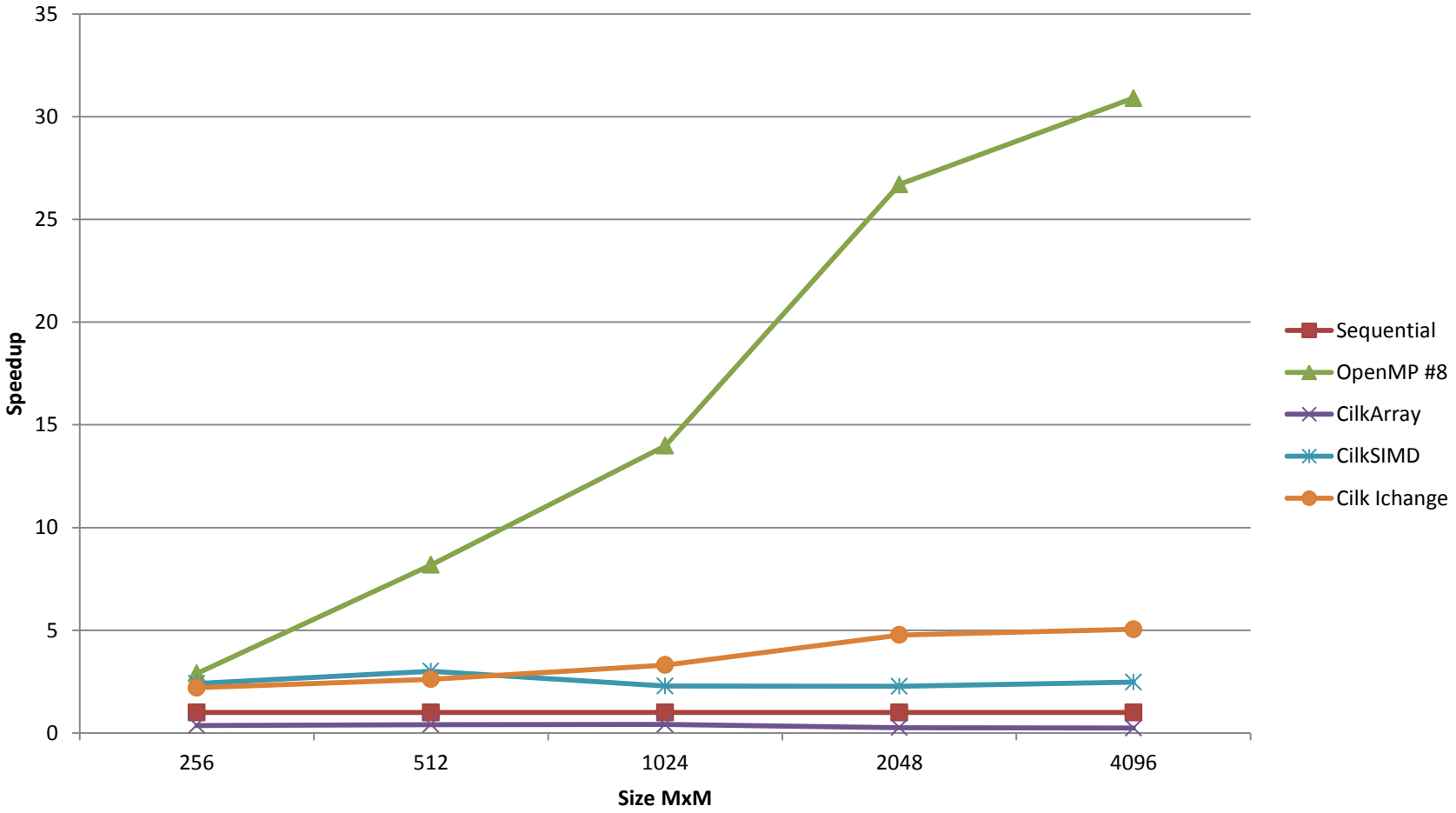
juan.jose.fumero.alfonso@cern.ch

Bibliography

1. A Guide to Auto-vectorization with Intel® C++ Compilers. Intel Documentation.
2. John L. Hennesy, David A. Patterson. *Computer Architecture. Quantitative Approach*. Fifth Edition (The Morgan Kaufmann Series in Computer Architecture Design).
3. Intel CilkPlus Documentation
4. Michael McCool, James Reinders, Arch Robinson. Structured Parallel Programming: Patterns for Efficient Computation. MK 2013.


CERN

Speedup – CilkPlus MxM



Intel(R) Xeon(R) CPU E3-1285L v3 @ 3.10GHz

GEMM (CEAN and CilkFor)