Práctica 1. Vectorización, paralelismo de datos

Computación de Altas Prestaciones

Carlos García Sánchez

19 de septiembre de 2022

- "Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition", James Jeffers, James Reinders, Avinash Sodani
- Vídeo YouTube: https://www.youtube.com/watch?v=Vwt1IKtkdI0

Outline

Equipo del laboratorio

- Modelo del procesador
- Número de cores
- Vectorización: *sse*, *avx*...
- Especificaciones a consultar en la página de Intel[1]

Terminal #1

```
user@lab:~ $  more /proc/cpuinfo
...
processor   : 3
...
model name  : Intel(R) Xeon(R) CPU E3-1225 v3 @ 3.20GHz
...
flags       : ... avx ... avx2 ...
....
```

[1] E... ...

Equipo del laboratorio

Compiladores

- GNU-GCC: **gcc** https://gcc.gnu.org
- Intel Compiler: **icc/icx** https://software.intel.com/en-us/c-compilers
- PGI Compilers: **pgcc** https://www.pgroup.com

Variables de entorno

Compilador de Intel

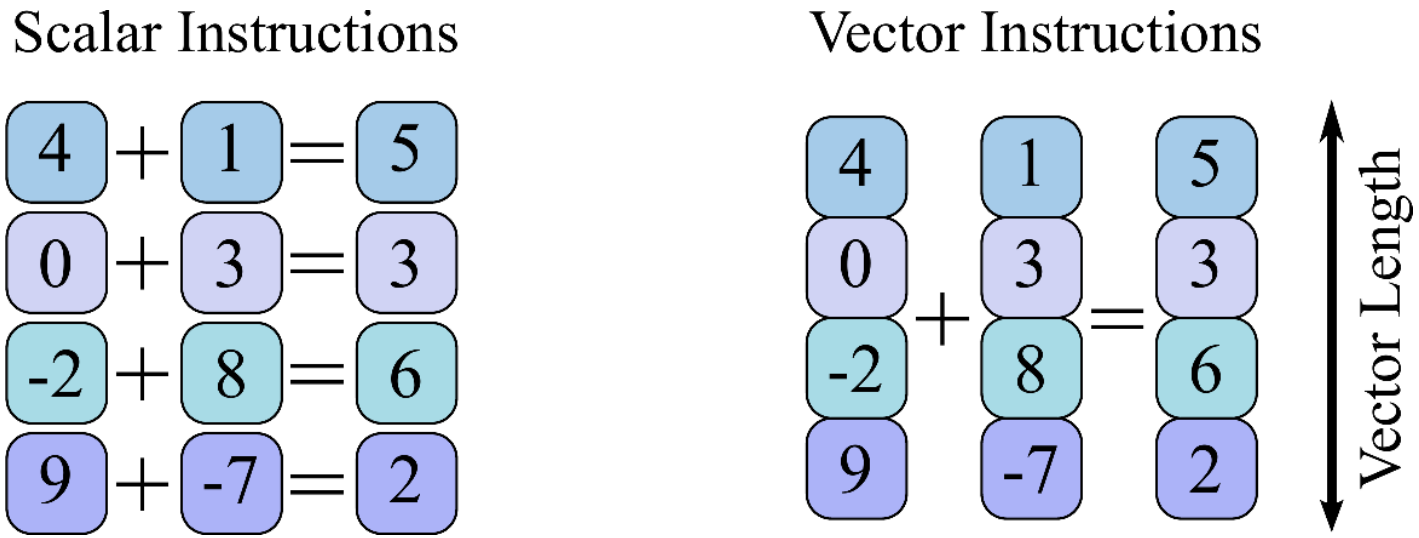
- Por defecto están incluidas en el .bashrc
  - Compilador de Intel (ICC)
  - Herramienta de Monitorización (ADVISOR)
  - Paralelismo con paso de mensajes (MPI)

Terminal #1

```
user@lab:~ $ source /opt/intel/oneapi/setvars.sh
:: initializing oneAPI environment ...
   bash: BASH_VERSION = 4.4.20(1)-release
:: advisor -- latest
:: ccl -- latest
:: clck -- latest
:: compiler -- latest
:: dal -- latest
:: debugger -- latest
:: dev-utilities -- latest
:: dnnl -- latest
:: dpcpp-ct -- latest
:: dpl -- latest
:: inspector -- latest
:: intelpython -- latest
:: ipp -- latest
:: ippcp -- latest
:: ipp -- latest
:: itac -- latest
:: mkl -- latest
:: mpi -- latest
:: tbb -- latest
:: vpl -- latest
:: vtune -- latest
:: oneAPI environment initialized ::
```

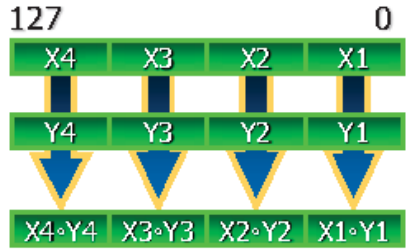
Soporte operaciones vectoriales

- Pequeños vectores = SIMD
- SIMD (Single Instruction Multiple Data)



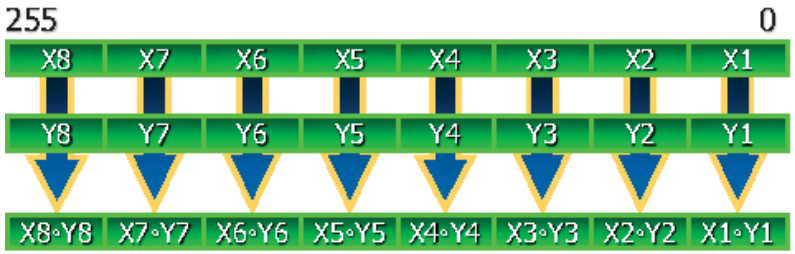
Soporte operaciones vectoriales

- Introducción en el ISA de Intel a partir de 1998



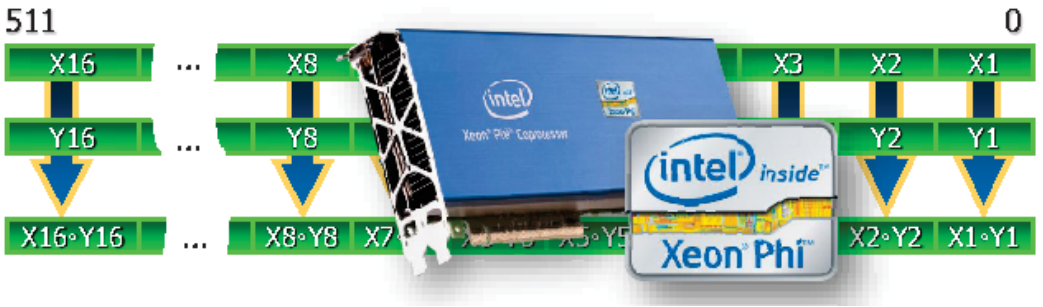
SSE
Vector size: **128 bit**
Data types:
- 8, 16, 32, 64 bit integer
- 32 and 64 bit float

VL: 2, 4, 8, 16



AVX
Vector size: **256 bit**
Data types:
- 8, 16, 32, 64 bit integer
- 32 and 64 bit float

VL: 4, 8, 16, 32



Intel® MIC
Vector size: **512 bit**
Data types:
- 32 bit integer
- 32 and 64 bit float

VL: 8, 16

¿Como vectorizar? Tres aproximaciones

- Vectorización automática

vectorAdd_auto.c

```
double a[vec_width], b[vec_width], c[vec_width];
//...
for (i = 0; i < n; i++)
    c[i] = a[i] + b[i];
```

- Vectorización explícita (intrínsecas)

vectorAdd_instrisic.c

```
double a[8], b[8], c[8];
//...
__m512d A_v = _mm512_load_pd(a);
__m512d B_v = _mm512_load_pd(b);
__m512d C_v = _mm512_add_pd(A_v,B_v);
_mm512_store_pd(c, C_v);
```

- Vectorización guiada (#pragmas)

Opciones de compilación

- Especifica las optimizaciones en compilación[2]: -O[n]
  - Por defecto el compilador utilizar -**O2**

O0	Deshabilita cualquier optimización
O1	Habilita optimizaciones que incrementan tamaño código pero lo acelera Optimización global: analisis de flujo, análisis de vida de variables Deshabilita inlining y uso de intrinsecas
O2	**Activa vectorización** **Optimización a nivel de bucle**: distribución, predicado, intercambios... **Optimización entre funciones**: inlining, propagación constantes, sustitución hacia adelante, propagación de atributos, eliminación de la función sin cómputo... **Mejora rendimiento**: propagación constantes, propagación copia, eliminaciń de cómputo inútil, asignación de registros globales, especulación de control (saltos), desenrrollado bucles, renomabrado variables, optimizaciones de pila...
O3	Opt. más agresiva: fusión de bucles, unroll&jam Adecuada para código basados en bucles y cómputo fp

[2] https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference-o

Contenidos

- Opciones de optimización
- Autovectorización
- Optimización con memoria
- Cálculo de potencial eléctrico
- **nbody**
- **Black-Scholes**
- **Stencil (intrínsecas)**
- Código disponible en el repo de la asignatura

Extra

- Herramienta de Intel Advisor
- Otras herramientas: gprof, valgrind

Códigos

- Se puede hacer un clone del repositorio al completo para comenzar con el laboratorio

Terminal #1

```
user@lab:~ $ git clone https://github.com/garsanca/CAP/
Cloning into CAP...
remote: Enumerating objects: 170, done.
remote: Counting objects: 100% (170/170), done.
remote: Compressing objects: 100% (110/110), done.
remote: Total 170 (delta 46), reused 169 (delta 45), pack-reused 0
Receiving objects: 100% (170/170), 20.31 MiB | 4.55 MiB/s, done.
Resolving deltas: 100% (46/46), done.
```

Opciones de compilación

- Opciones de compilación (O0, O1, O2)

ejemplo1.c

```
void foo( float* sth, float* theta, int p) {

    for(int i = 0; i < p; i++)
        sth[i] = sin(theta[i]+3.1415927);
}
```

Terminal #1

```
user@lab:~ $ icc -o foo00 foo00.c -O0 -qopt-report=3 -xhost -lm
user@lab:~ $ more foo00.optrpt
Intel(R) Advisor can now assist with vectorization and show optimization
  report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.

Intel(R) C Intel(R) 64 Compiler for applications running on Intel(R) 64, Version 19.0.4.235 Build

Compiler options: -o foo00 -O0 -qopt-report=3 -xhost -lm
```

Autovectorización (O2)

Terminal #1

```
user@lab:~ $ icc -o foo02 foo02.c -O2 -qopt-report=3 -xhost -lm
user@lab:~ $ more foo02.optrpt
Intel(R) Advisor can now assist with vectorization and show optimization
  report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.

Intel(R) C Intel(R) 64 Compiler for applications running on Intel(R) 64, Version 19.0.4.235 Build

Compiler options: -o foo02 -O2 -qopt-report=3 -xhost -lm

    Report from: Interprocedural optimizations [ipo]

Intel(R) Advisor can now assist with vectorization and show optimization
  report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.

Intel(R) C Intel(R) 64 Compiler for applications running on Intel(R) 64, Version 19.0.4.235 Build

Compiler options: -o foo02 -O2 -qopt-report=3 -xhost -lm

...

LOOP BEGIN at foo00.c(23,2) inlined into foo00.c(58,2)
   remark #15300: LOOP WAS VECTORIZED
   remark #15442: entire loop may be executed in remainder
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15451: unmasked unaligned unit stride stores: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 111
   remark #15477: vector cost: 20.750
   remark #15478: estimated potential speedup: 4.780
   remark #15482: vectorized math library calls: 1
   remark #15487: type converts: 2
   remark #15488: --- end vector cost summary ---
LOOP END

...
```

Autovectorización (O2)

- A tener en cuenta
  - **Bucle vectorizado**
  - 1 load y 1 store **unmasked unaligned unit stride**
  - Tipo de conversion: **type converts: 2**
  - Resumen: **estimated potential speedup: 4.780**

ejemplo1.c

```
void foo( float* sth, float* theta, int p) {

   for(int i = 0; i < p; i++)
      sth[i] = sin(theta[i]+3.1415927);
}
```

Terminal #1

```
LOOP BEGIN at foo00.c(23,2) inlined into foo00.c(58,2)
   remark #15300: LOOP WAS VECTORIZED
   remark #15442: entire loop may be executed in remainder
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15451: unmasked unaligned unit stride stores: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 111
   remark #15477: vector cost: 20.750
   remark #15478: estimated potential speedup: 4.780
   remark #15482: vectorized math library calls: 1
   remark #15487: type converts: 2
   remark #15488: --- end vector cost summary ---
LOOP END
...
```

Autovectorización (O2)

- A tener en cuenta
  - **Bucle vectorizado**
  - 1 load y 1 store **unmasked unaligned unit stride**
  - Tipo de conversion: **No hay**
  - Resumen: **estimated potential speedup: 7.780**

ejemplo1.c

```
void foo( float* sth, float* theta, int p) {

    for(int i = 0; i < p; i++)
        sth[i] = sin(theta[i]+3.1415927f);
}
```

Terminal #1

```
...
LOOP BEGIN at foo02_conversion.c(23,2) inlined into foo02_conversion.c(58,2)
   remark #15300: LOOP WAS VECTORIZED
   remark #15442: entire loop may be executed in remainder
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15451: unmasked unaligned unit stride stores: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 109
   remark #15477: vector cost: 10.000
   remark #15478: estimated potential speedup: 7.780
   remark #15482: vectorized math library calls: 1
   remark #15488: --- end vector cost summary ---
LOOP END
...
```

Autovectorización (O2)

- Diferencia compilar para SSE y AVX: **-xCORE_AVX2** **-xSSE4.2**

Terminal #1

```
LOOP BEGIN at foo02_conversion.c(23,2) inlined into foo02_conversion.c(58,2)
   remark #15300: LOOP WAS VECTORIZED
   remark #15442: entire loop may be executed in remainder
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15451: unmasked unaligned unit stride stores: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 109
   remark #15477: vector cost: 10.000
   remark #15478: estimated potential speedup: 7.780
   remark #15482: vectorized math library calls: 1
   remark #15488: --- end vector cost summary ---
LOOP END

...
```

Terminal #1

```
LOOP BEGIN at foo02_conversion.c(23,2) inlined into foo02_conversion.c(58,2)
   remark #15300: LOOP WAS VECTORIZED
   remark #15442: entire loop may be executed in remainder
   remark #15448: unmasked aligned unit stride loads: 1
   remark #15449: unmasked aligned unit stride stores: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 109
   remark #15477: vector cost: 19.750
   remark #15478: estimated potential speedup: 5.190
   remark #15482: vectorized math library calls: 1
   remark #15488: --- end vector cost summary ---
LOOP END

...
```

alineamiento memoria

- A tener en cuenta
  - **Bucle vectorizado**
  - 1 load y 1 store **unmasked aligned unit stride loads**
  - Tipo de conversion: **no hay**
  - Resumen: **estimated potential speedup: 9.730**

ejemplo1_aligned.c

```
...
   sth  = (float*)_mm_malloc(n*sizeof(float), 16);
   theta = (float*)_mm_malloc(n*sizeof(float), 16);
...

void foo( float* sth, float* theta, int p) {
   #pragma vector aligned
   for(int i = 0; i < p; i++)
      sth[i] = sin(theta[i]+3.1415927f);
}
```

Terminal #1

```
LOOP BEGIN at foo02_aligned.c(24,2) inlined into foo02_aligned.c(59,2)
   remark #15300: LOOP WAS VECTORIZED
   remark #15448: unmasked aligned unit stride loads: 1
   remark #15449: unmasked aligned unit stride stores: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 109
   remark #15477: vector cost: 9.870
   remark #15478: estimated potential speedup: 9.730
   remark #15482: vectorized math library calls: 1
   remark #15488: --- end vector cost summary ---
LOOP END

...
```

Cálculo potencial eléctrico

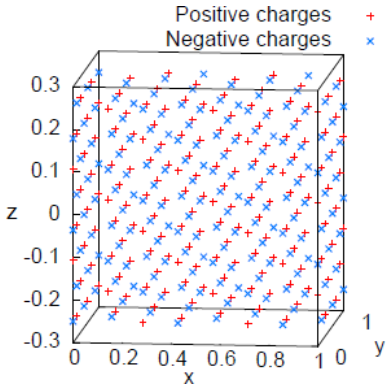
- Consideramos a distribución de un conjunto de m partículas posicionados en $\vec{r} = (r_{i,x}, r_{i,y}, r_{i,z})$ y con cargas q_i
- Si queremos calcular el potencial eléctrico $\phi\left(\vec{R}\right)$ en múltiples localizaciones $\vec{R_j} = (R_{j,x}, R_{j,y}, R_{j,z})$ donde j denota una de las localizaciones de n

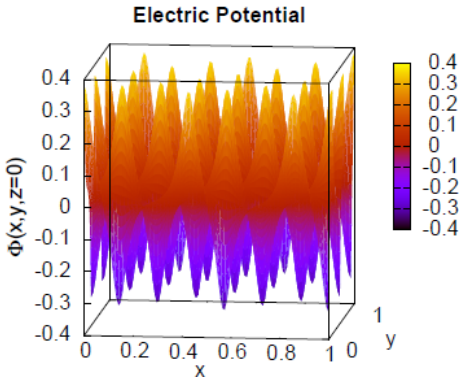
coulomb.c

```
struct Charge {// Elegant, but ineffective data layout
     float x, y, z, q; // Coordinates and value of this charge
     int type; // Positive or negative charge
};
ChargeType* particle; // Now declare an array of m point charges
```

Cálculo potencial eléctrico

- Cálculo de potencial eléctrico
  - Enlace al libro extraido!
  - Enlace al github!





$$\phi\left(\vec{R}\right) = -\sum_{i=1}^{m} \frac{q_i}{\sqrt{(r_{i,x} - R_{j,x})^2 + (r_{i,y} - R_{j,y})^2 + (r_{i,z} - R_{j,z})^2}} \quad (1)$$

Cálculo potencial eléctrico

- Versión 0:

coulomb.c

```
coulomb.c:
// This version performs poorly, because data layout of class Charge
// does not allow efficient vectorization
void CalculateElectricPotential(
       const int m,    // Number of charges
       const Charge* chg, // Charge distribution (array of classes)
       const double Rx, const double Ry, const double Rz, // Observation
            point
       float & phi // Output: electric potential
       ) {
   phi=0.0f;
   for (int i=0; i<m; i++) { // This loop will be auto-vectorized
       // Non-unit stride: (&chg[i+1].x - &chg[i].x) == sizeof(Charge)
       const double dx=chg[i].x - Rx;
       const double dy=chg[i].y - Ry;
       const double dz=chg[i].z - Rz;
       phi -= chg[i].q / sqrt(dx*dx+dy*dy+dz*dz); // Coulomb's law
   }
}
```

Cálculo potencial eléctrico

- Generando el informe: **make REPORT=yes**

Terminal #1

```
user@lab:~ $ make REPORT=yes
user@lab:~ $ icpc -g -std=c++11 -O2 -I.  -qopt-report=5   -c -o main.o main.cc
   icpc: remark #10397: optimization reports are generated in *.optrpt files in the output locatio
```

- Como de detallado es el informe: **264 lines** (en total, solo main.optrpt 264)
- Filtrando el report: **make REPORT=yes FILTER=yes**

Terminal #1

```
user@lab:~ $ make REPORT=yes FILTER=yes
user@lab:~ $ icpc -g -std=c++11 -O2 -I.  -qopt-report=5 -qopt-report-phase=vec -qopt-report-
filter="main.cc,15-25"   -c -o main.o main.cc
```

Cálculo potencial eléctrico

Terminal #1

```
user@lab:~ $ more main.optrpt
...
LOOP BEGIN at main.cc(20,5) inlined into main.cc(74,13)
remark #15305: vectorization support: vector length 2
        remark #15309: vectorization support: normalized vectorization overhead 0.367
        ...
        remark #15355: vectorization support: *phi is float type reduction   [ main.cc(25,9) ]
        remark #15300: LOOP WAS VECTORIZED
        remark #15452: unmasked strided loads: 4
        remark #15475: --- begin vector cost summary ---
        remark #15476: scalar cost: 82
        remark #15477: vector cost: 39.500
        remark #15478: estimated potential speedup: 2.070
        remark #15487: type converts: 6
        remark #15488: --- end vector cost summary ---
LOOP END
...
```

Cálculo potencial eléctrico

- ¿Hay algún cuello de botella?
- ¿Cuál es el máximo rendimiento?
- ¿Tiene algún problema el código?

Terminal #1

```
user@lab:~ $ ./coulomb
Initialization... complete.
Trial    Time, s  GFLOP/s
    1  4.497e+00      2.4 *
    2  4.556e+00      2.4 *
    3  4.594e+00      2.3
    4  4.330e+00      2.5
    5  3.853e+00      2.8
    6  3.855e+00      2.8
    7  3.841e+00      2.8
    8  3.832e+00      2.8
    9  3.845e+00      2.8
   10  3.842e+00      2.8
-----------------------------------------------------
Average performance:            2.7 +- 0.2 GFLOP/s
-----------------------------------------------------
* - warm-up, not included in average
```

Cálculo potencial eléctrico

- Report: **main.optrpt**
  - **LOOP WAS VECTORIZED**: vector length 2
  - **estimated potential speedup: 2.070**
  - **type converts: 6**

Terminal #1

```
    LOOP BEGIN at main.cc(20,5) inlined into main.cc(74,13)
remark #15305: vectorization support: vector length 2
        remark #15309: vectorization support: normalized vectorization overhead 0.367
        ....
        remark #15355: vectorization support: *phi is float type reduction   [ main.cc(25,9) ]
        remark #15300: LOOP WAS VECTORIZED
        remark #15452: unmasked strided loads: 4
        remark #15475: --- begin vector cost summary ---
        remark #15476: scalar cost: 82
        remark #15477: vector cost: 39.500
        remark #15478: estimated potential speedup: 2.070
        remark #15487: type converts: 6
        remark #15488: --- end vector cost summary ---
      LOOP END
```

Cálculo potencial eléctrico

- ¿Como poder mejorar el rendimiento usando el compilador?
  - Se pueden emplear diferentes opciones de compilación y probar con otras arquitecturas: **-xCORE-AVX2**

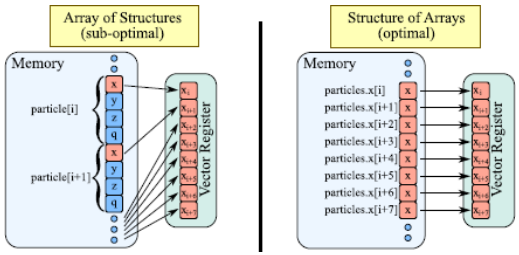
Cálculo potencial eléctrico

- **Version 2**: El compilador genera instrs. gather/scatter facilitando la vectorización para el caso de acceso no contiguo (vectorización)

coulomb.c

```
struct Charge {// Elegant, but ineffective data layout
     float x, y, z, q; // Coordinates and value of this charge
     int type; // Positive or negative charge
};
ChargeType* particle; // Now declare an array of m point charges


struct Charge_Distribution {
   // This data layout permits effective vectorization of Coulomb's law
         application
   const int m; // Number of charges
   float * x; // Array of x-coordinates of charges
   float * y; // ...y-coordinates...
   float * z; // ...etc.
   float * q; // Charges
   int * type; // Positive or negative charge
};
```



Cálculo potencial eléctrico

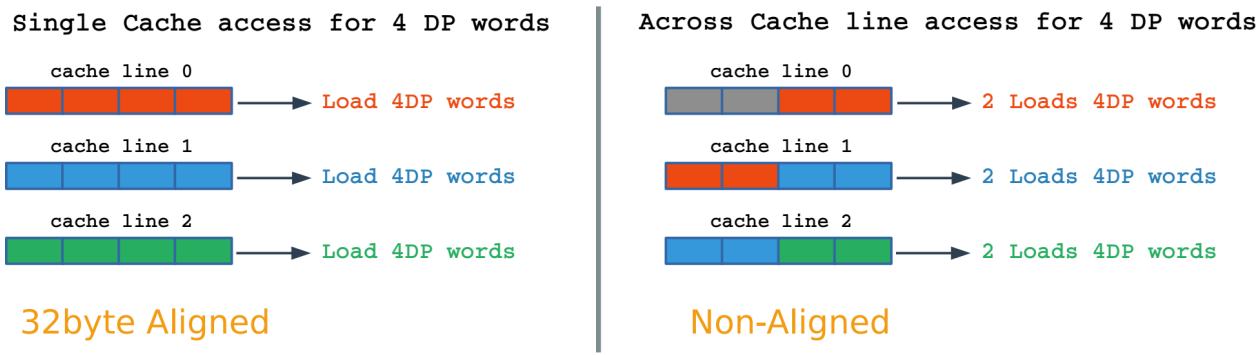
- **Version 2**: evaluación de AoS vs SoA
  - **estimated potential speedup: 6.570**

Terminal #1

```
LOOP BEGIN at main.cc(23,5) inlined into main.cc(81,13)
     remark #15389: vectorization support: reference chg.x[i] has unaligned access  [main.cc(2
     remark #15381: vectorization support: unaligned access used inside loop body
     remark #15305: vectorization support: vector length 8
     remark #15309: vectorization support: normalized vectorization overhead 0.812
     remark #15300: LOOP WAS VECTORIZED
     remark #15442: entire loop may be executed in remainder
     remark #15450: unmasked unaligned unit stride loads: 4
     remark #15475: --- begin vector cost summary ---
     remark #15476: scalar cost: 70
     remark #15477: vector cost: 8.000
     remark #15478: estimated potential speedup: 6.570
     remark #15488: --- end vector cost summary ---
  LOOP END
```

Cálculo potencial eléctrico

- **Versión 3**: Alineamiento de memoria
  - El compilador no puede saber si los datos están alineados con un múltiplo del ancho del registro vectorial. Esto podría afectar el rendimiento.
  - El puntero p está alineado en memoria a n-byte si: ((size_t)p %n == 0)
  - En **AVX**, el alineamiento se produce a **32bytes** (4DP words): un único acceso a línea de cache permite mover 4 DP words a registros vectoriales



Cálculo potencial eléctrico

- En la pila: variables declaradas son alineadas de forma natural en el Intel® C/C++:
  - `float f; //4-byte aligned`
  - `double d; //8-byte aligned`
- En el caso de arrays de datos es necesario especificarlo con un atributo:
  - `float array[N] __attribute__((aligned(32)); //32-byte aligned`

Cálculo potencial eléctrico

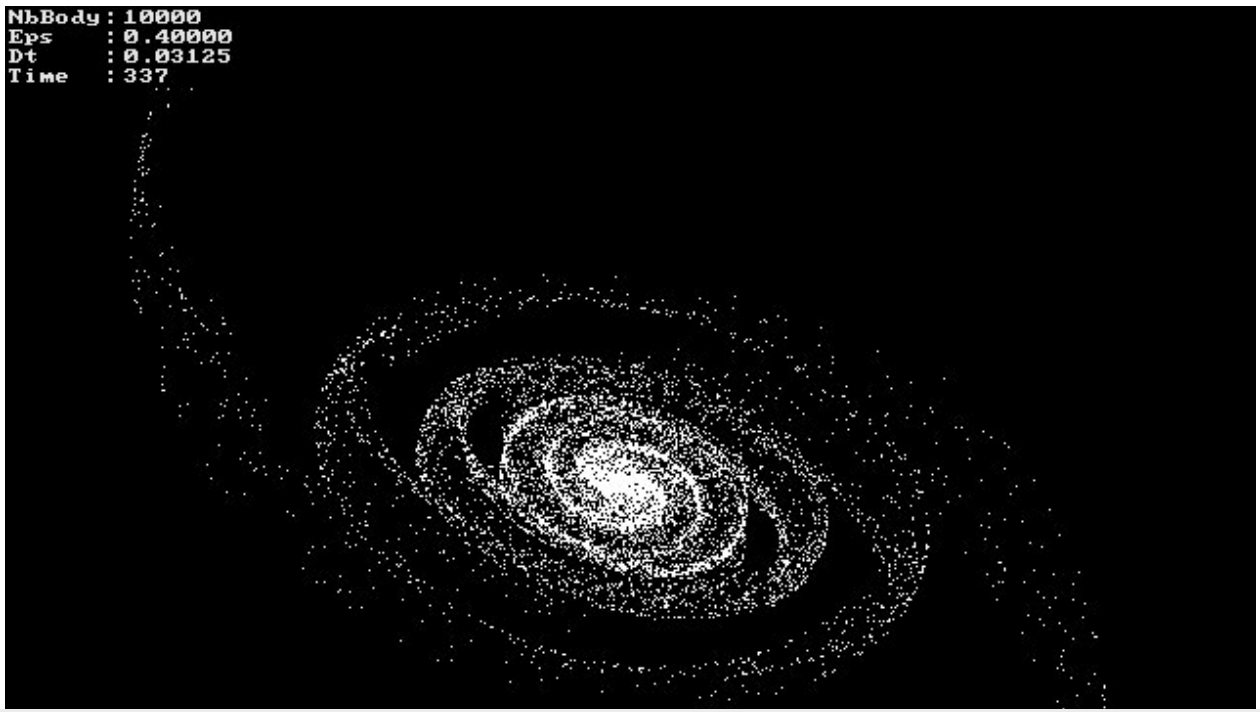
- En el Heap: el array puede ser reservado/liberado con funciones malloc/free especiales:

memoryAligned.c

```
#include <malloc.h>
...
float *array = (float*) _mm_malloc(N*sizeof(float), 32);
...
_mm_free(array);
```

N-Body

- En astrofísica se usa para resolver problema de la predicción de los movimientos individuales de un grupo de objetos celestes que interactúan entre sí gravitacionalmente
  - Ej: sistemas tierra-luna-sol



N-Body

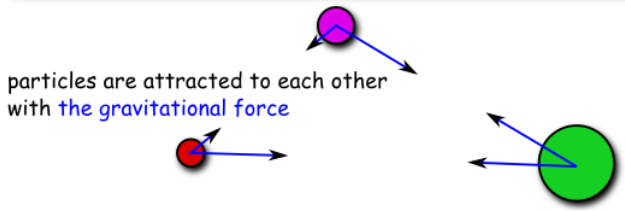
Gravitational N-body dynamics:

Newton's law of universal gravitation:

$$M_i \vec{R}_i''(t) = G \sum_j \frac{M_i M_j}{\left|\vec{R}_i - \vec{R}_j\right|^3} \left(\vec{R}_j - \vec{R}_i\right)$$

where:

$$\left|\vec{R}_i - \vec{R}_j\right| = \sqrt{(R_{i,x} - R_{j,x})^2 + (R_{i,y} - R_{j,y})^2 + (R_{i,z} - R_{j,z})^2}$$



Cálculo de fuerzas gravitatorias

$$F_{i,j} = m_i \frac{\partial^2 R_i}{\partial^2 t}$$

$$F_{i,j} = \frac{G m_i m_j \left(R_j - R_i\right)}{\left\|R_j - R_i\right\|^3}$$

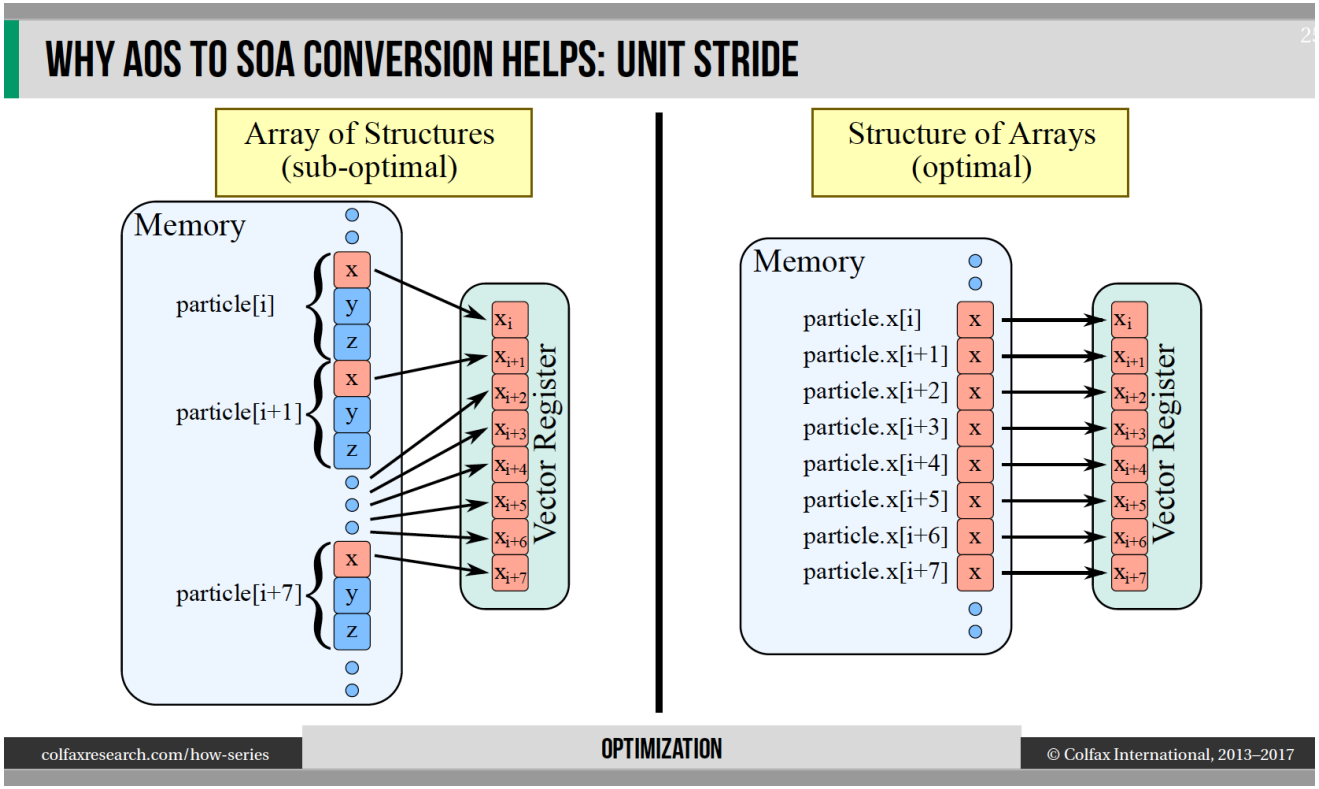
$$m_i \frac{\partial^2 R_i}{\partial^2 t} = \sum_{j=0, i \neq j}^{N} \frac{G m_i m_j \left(R_j - R_i\right)}{\left\|R_j - R_i\right\|^3} = \frac{\partial U}{\partial R_i}$$

N-Body

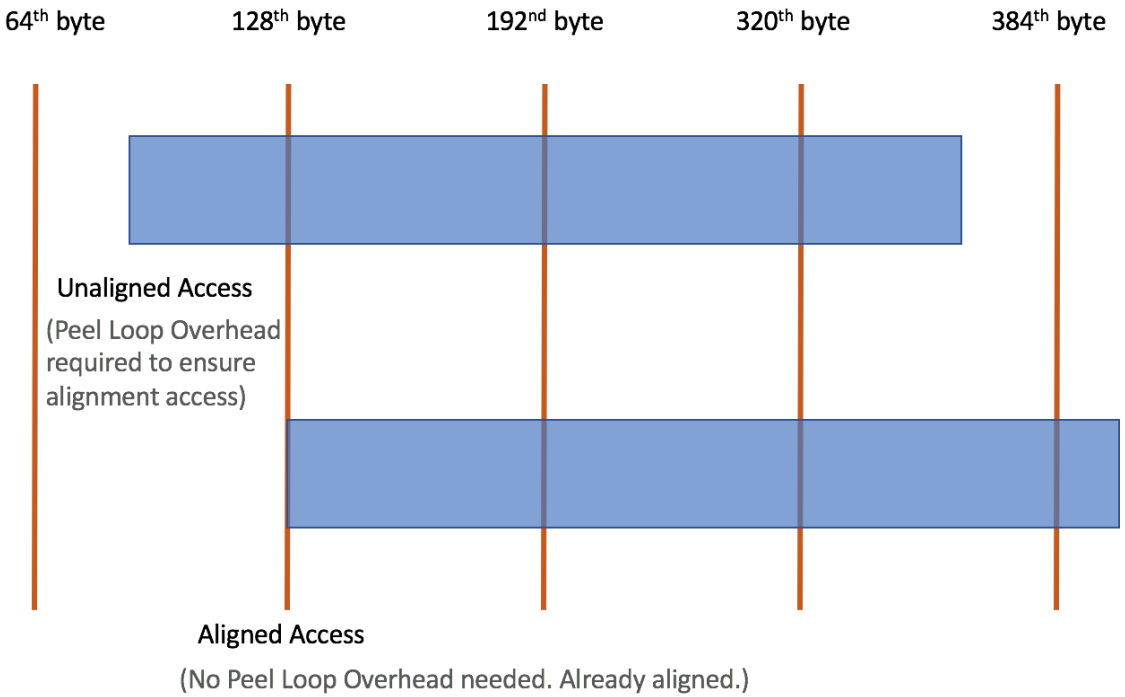
Optimizaciones a estudiar

- Versión original
- Vectorización autovectorizada
- Vectorización guiada
  - Eliminación de "dependencias" en bucles (desalineación de punteros o #pragma)
  - Eliminación de conversión de tipos de datos (float/double)
- Accesos a memoria (**¿stride?**): SoA vs AoS
- Memoria alineada

N-Body


WHY AOS TO SOA CONVERSION HELPS: UNIT STRIDE
Array of Structures (sub-optimal)
Structure of Arrays (optimal)
Memory
particle[i]
particle[i+1]
particle[i+7]
x
y
z
Vector Register
Memory
particle.x[i]
particle.x[i+1]
particle.x[i+2]
particle.x[i+3]
particle.x[i+4]
particle.x[i+5]
particle.x[i+6]
particle.x[i+7]
colfaxresearch.com/how-series
OPTIMIZATION
© Colfax International, 2013–2017

N-Body


64th byte
128th byte
192nd byte
320th byte
384th byte
Unaligned Access
(Peel Loop Overhead required to ensure alignment access)
Aligned Access
(No Peel Loop Overhead needed. Already aligned.)

Black-Scholes

- Modelo de Black-Scholes utilizado en matemática financiera para evaluar precio de activos (Fisher Black, Robert Merton and Myron Scholes en 1973)
  - Empleado para estimar el valor de compra (*call*) y venta (*put*) de acciones

Black-Scholes

$$Call = S\Phi d_1 - Ke^{-rT}\Phi d_2$$
$$Put = Ke^{-rT}\Phi(-d_2) - S\Phi(-d_1)$$
donde: $d_1 = \frac{ln(\frac{S}{K})+(r+\sigma^2/2)T}{\sigma\sqrt{T}}$ $d_2 = \frac{ln(\frac{S}{K})+(r-\sigma^2/2)T}{\sigma\sqrt{T}}$

Definiendo

- S: tasa a la vista de la moneda
- K: precio marcado en la opción (*strike price*)
- T: tiempo expresado en años
- r: es la tasa de interés
- σ: volatilidad
- Φ: distribución normal acumulada

Black-Scholes

Optimizaciones a estudiar

- Versión original
- Vectorización guiada (dependencias)
- Memoria alineada

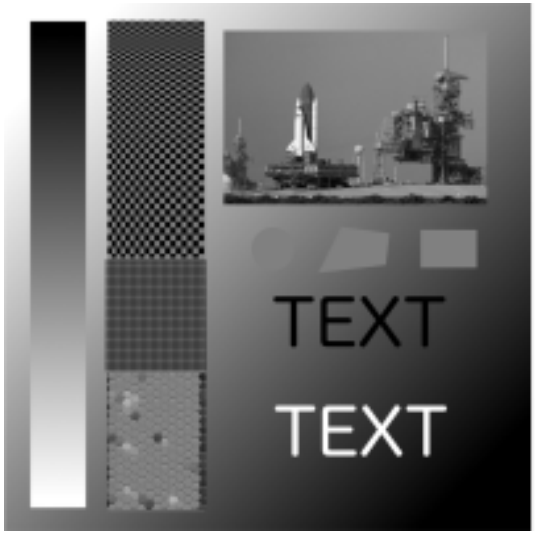
Intrinsecas

- La vectorización con intrínsecas es a veces la única forma de vectorizar un bucle de forma eficiente

Operador *stencil*

- Tratamiento de imágenes

Intrinsecas



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \rightarrow$$

Intrinsecas

stencil.c

```
void ApplyStencil(unsigned char *img_in, unsigned char *img_out, int
     width, int height) {

   short val;
   unsigned char val_out;

   for (int i = 1; i < height-1; i++)
      for (int j = 1; j < width-1; j++) {
         val = img_in[(i )*width + j];
         val += -img_in[(i-1)*width + j-1] - img_in[(i-1)*width + j] -
              img_in[(i-1)*width + j+1]
                -img_in[(i )*width + j-1] + 7*img_in[(i )*width + j] -
                     img_in[(i )*width + j+1]
                -img_in[(i+1)*width + j-1] - img_in[(i+1)*width + j] -
                     img_in[(i+1)*width + j+1];
         if (val<0){
            val=0;
         }else val_out = (unsigned char)val;
         if (val>255){
            val_out=255;
         }else val_out = (unsigned char)val;

         img_out[i*width + j] = (unsigned char)(val_out);
      }
}
```

Intrinsecas

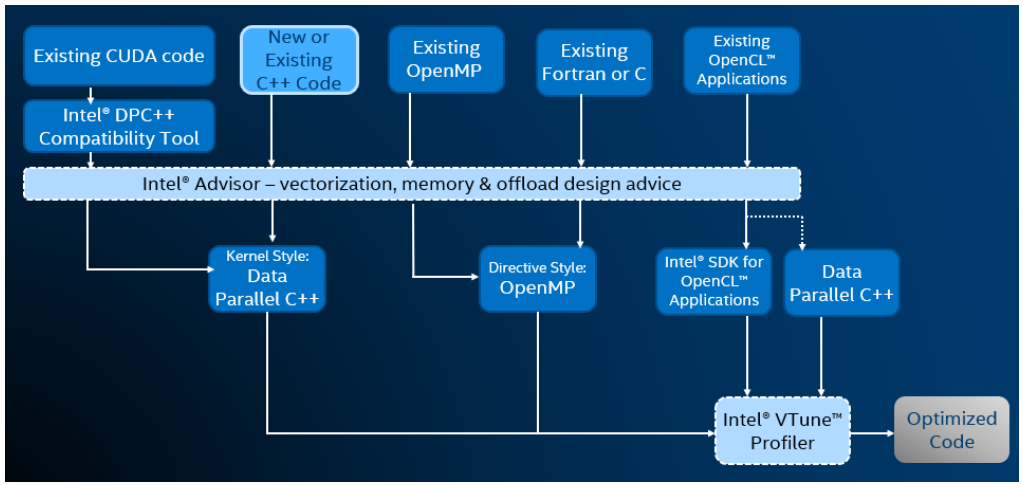
Optimizaciones a estudiar

- Versión original
- Vectorización guiada (dependencias) + memoria alineada
- Uso de intrínsecas [3]
  - Aritmética en entero (8 bits) con saturación
  - *Add/sub packed unsigned 8-bit integers using saturation*

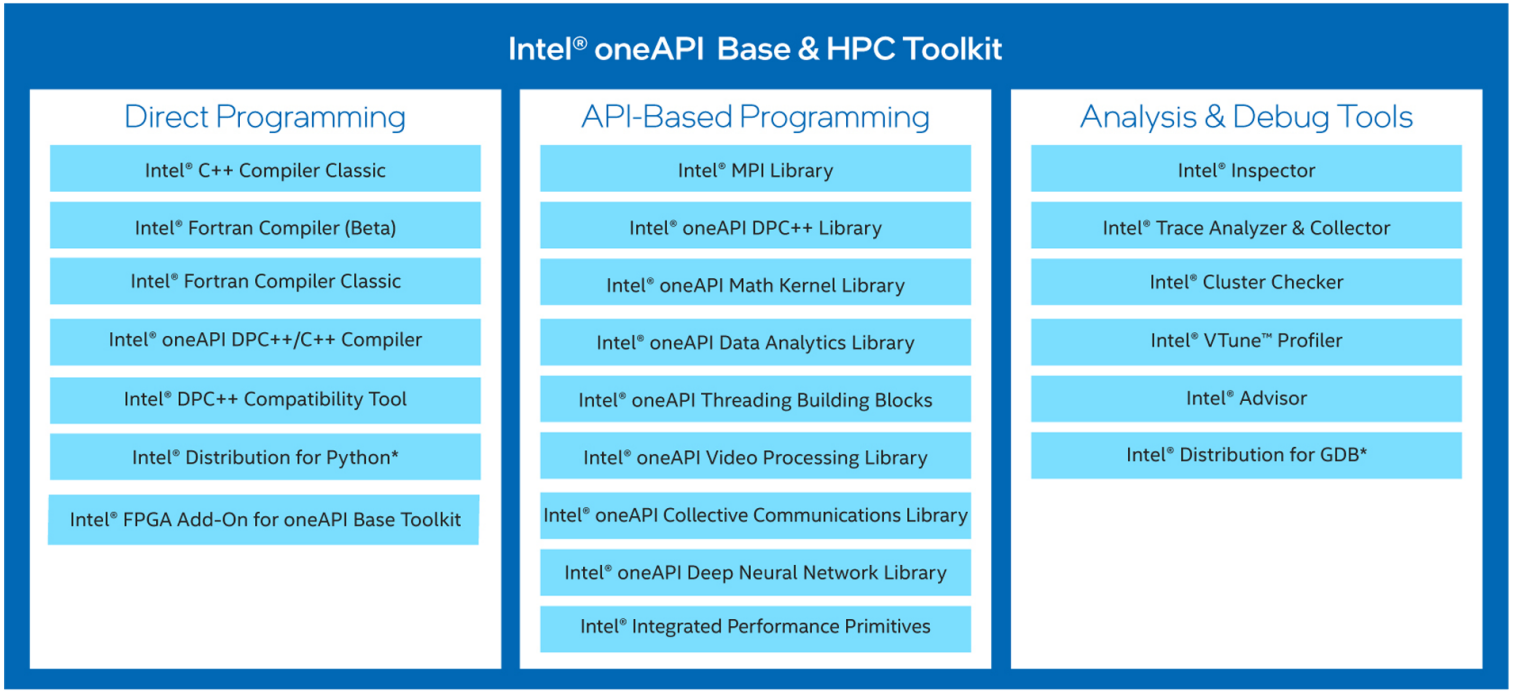
[3] The Intel® Intrinsics Guide:
https://software.intel.com/sites/landingpage/IntrinsicsGuide/

Intel Advisor

- Herramientas de perfilado como Intel® Advisor (vectorización y la memoria)



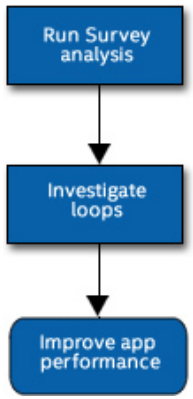
Intel Advisor



*Other names and brands may be claimed as the property of others.

Intel Advisor

- Lanzar el *advisor-gui* en un equipo local
- ... o recolectar la información *advixe-cl -collect survey -project-dir path-project ./exec*
- Opciones de compilación: *-O2 -g -xhost*
- Descarga el Intel-Advisor para Windows, MacOS o Linux[4]



[4] Intel Advisor https://drive.google.com/drive/u/0/folders/1Jw-nb3XmmoRO1zVVcisWNFtnWgDpHFBs

Intel Advisor

- Documentación en [5]
- Vídeo de demostración de funcionamiento y ejemplo de uso [6]

[5] Intel Advisor: https://software.intel.com/en-us/get-started-with-advisor

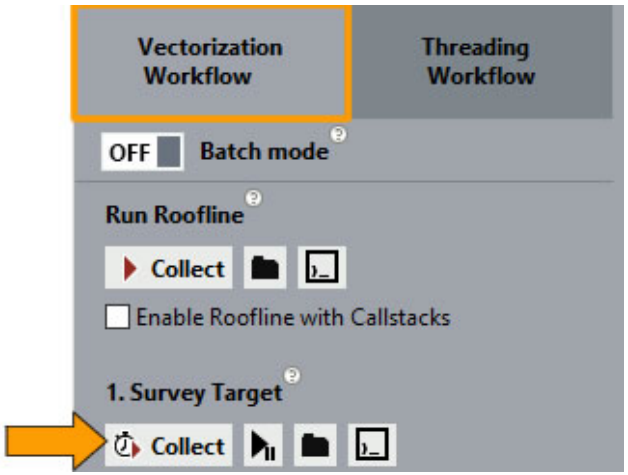
[6] Video Demo Intel Advisor: https://software.intel.com/en-us/videos/introduction-to-intel-advisor-vectorization-workflow

Intel Advisor (GUI)

- Crear proyecto
  - En **File** > **New** > **Project** se abre la ventana de diálogo. Suministrar nombre y localización (path) del proyecto, y por último pulsar *Create Project*

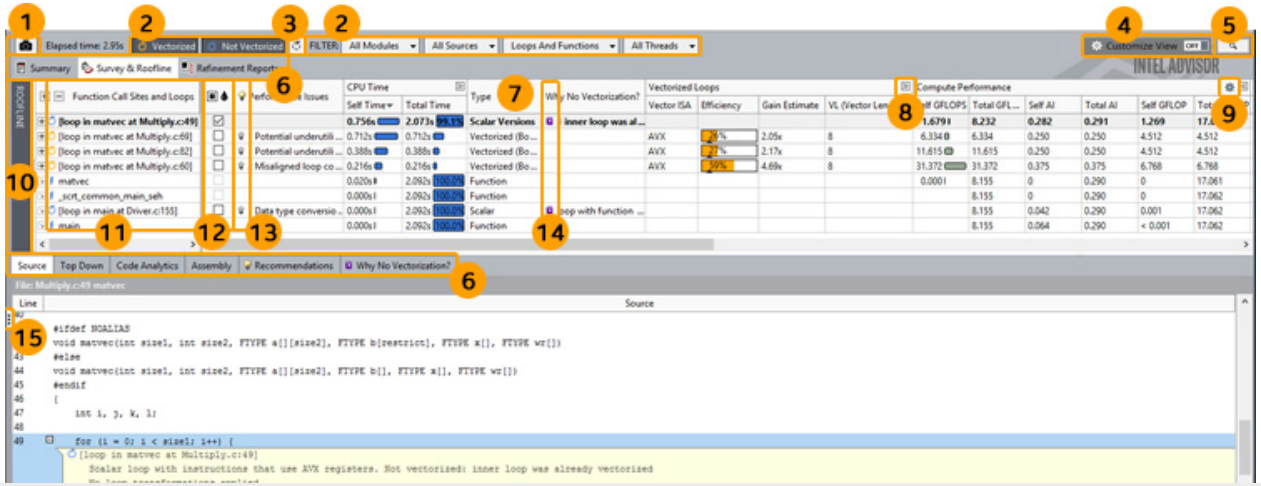
Intel Advisor (survey)

- Para recolectar la información pulsar el botón *collect* y con la pestaña *Vectorization Workflow* activada se obtiene un análisis del código



Intel Advisor (survey)

- **1** Botón de control para guardar resultados
- **2** y **3** Filtro de resultados
- **10** Modelo roofline
- **11** Resumen: funciones vectorizadas, bucles vectorizados, función escalar y para bucles escalares
- **13** Panel de recomendaciones (Optimización)
- **14** Razones para la no vectorización

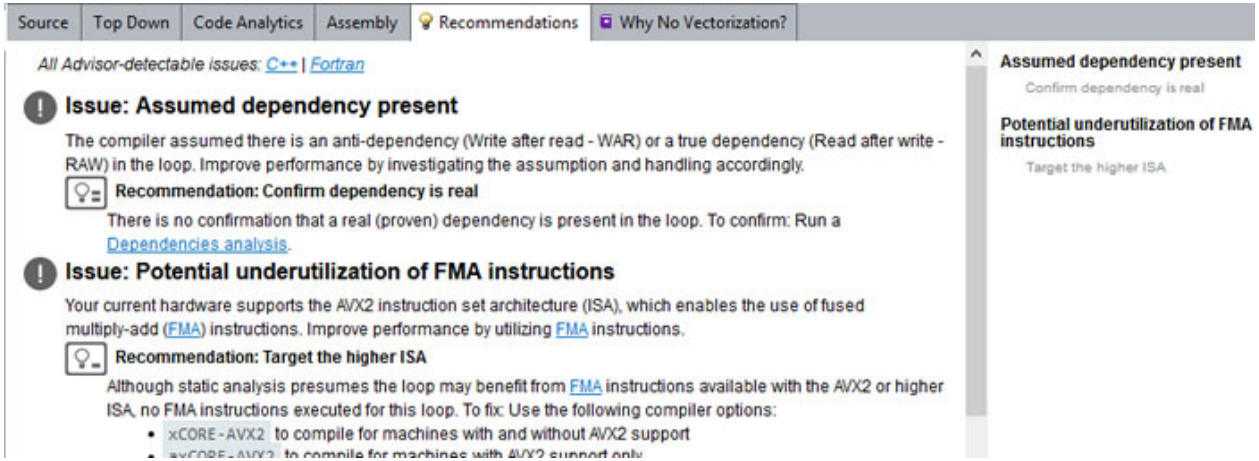


Intel Advisor (recommendation pannel)

- Si todos los bucles son vectorizados apropiadamente (>90 % eficiencia): **rendimiento satisfactorio**
- Si no... En *Performance Issues* y panel *Recommendations* aparecen una serie de recomendaciones:
  - **¿Por qué no se vectoriza?** Posibles dependencias, llamada a función, o que simplemente el compilador lo determina como ineficiente [7]
  - Si **vectoriza pero de forma ineficiente**: patrón de acceso a memoria ineficiente, tipo de conversiones de datos, falta de acceso a memoria alineado...

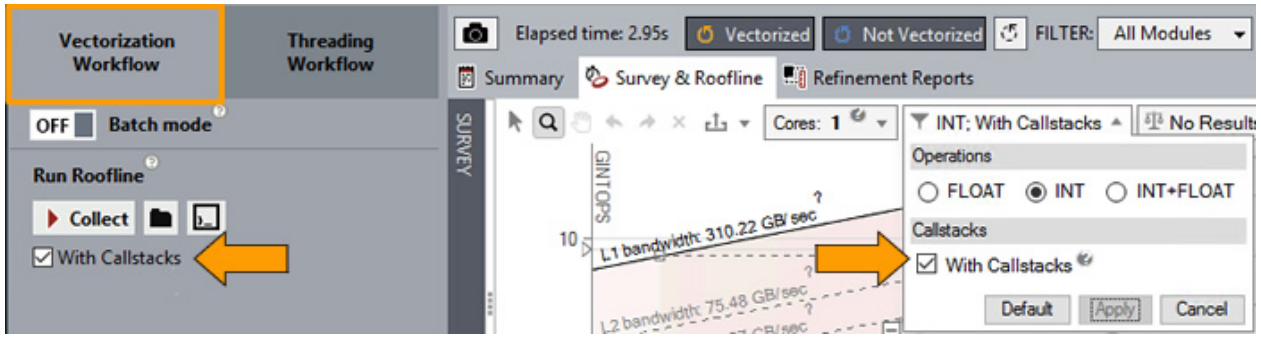
[7] https://software.intel.com/en-us/advisor-user-guide-next-steps-after-running-survey-analysis

Intel Advisor (recommendation pannel)



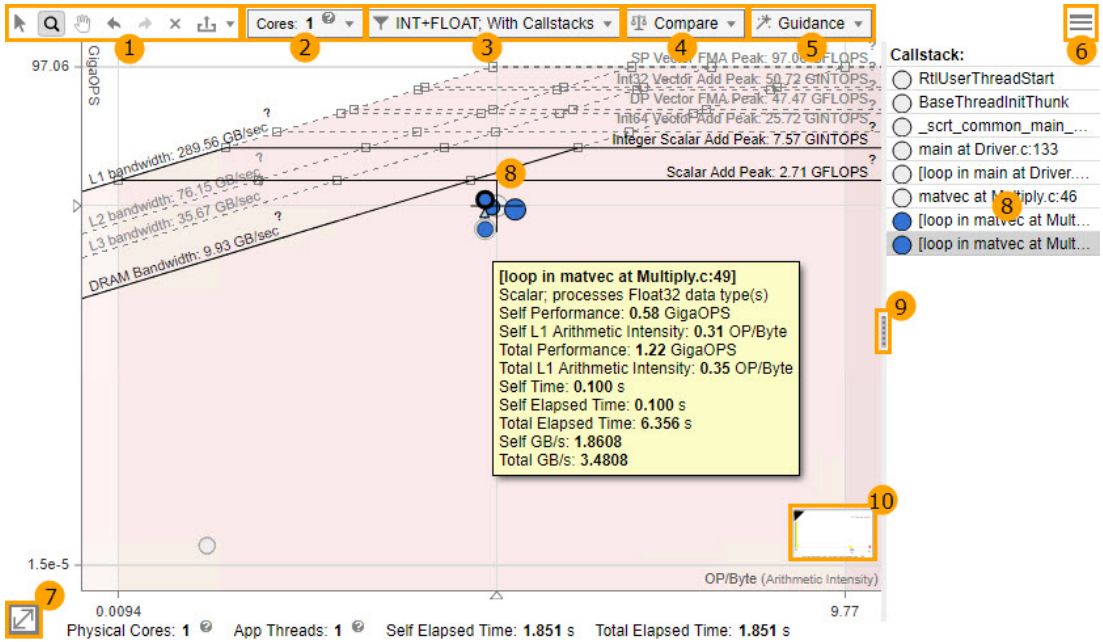
Intel Advisor (roofline)

- Permite obtener modelo roofline
- Detectar posibles problemas (*Code Analysis*)



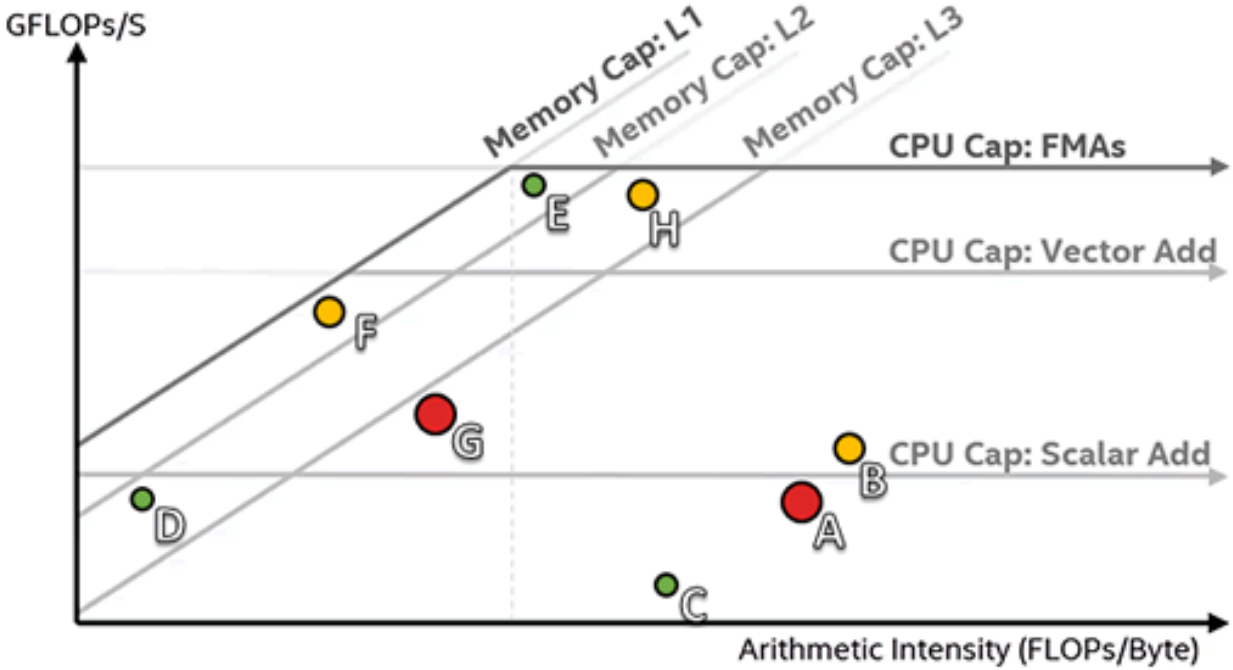
Intel Advisor (roofline)

- **2** Variación del número de cores para el análisis (TLP)
- **3** Análisis para operaciones FLOAT, INT o ambas
- **5** Otros indicadores interesantes del modelo roofline: jerarquía de memoria (*Show memory level relationships*)



Intel Advisor (roofline)

- Bucles A y G (puntos rojos) y en menor medida B (amarillo), son candidatos a **paralelizar**
- Bucles C, D y E (puntos verdes) y H (amarillo) son malos candidatos porque tiene poco impacto en el rendimiento global



Intel Advisor (Memory Access)

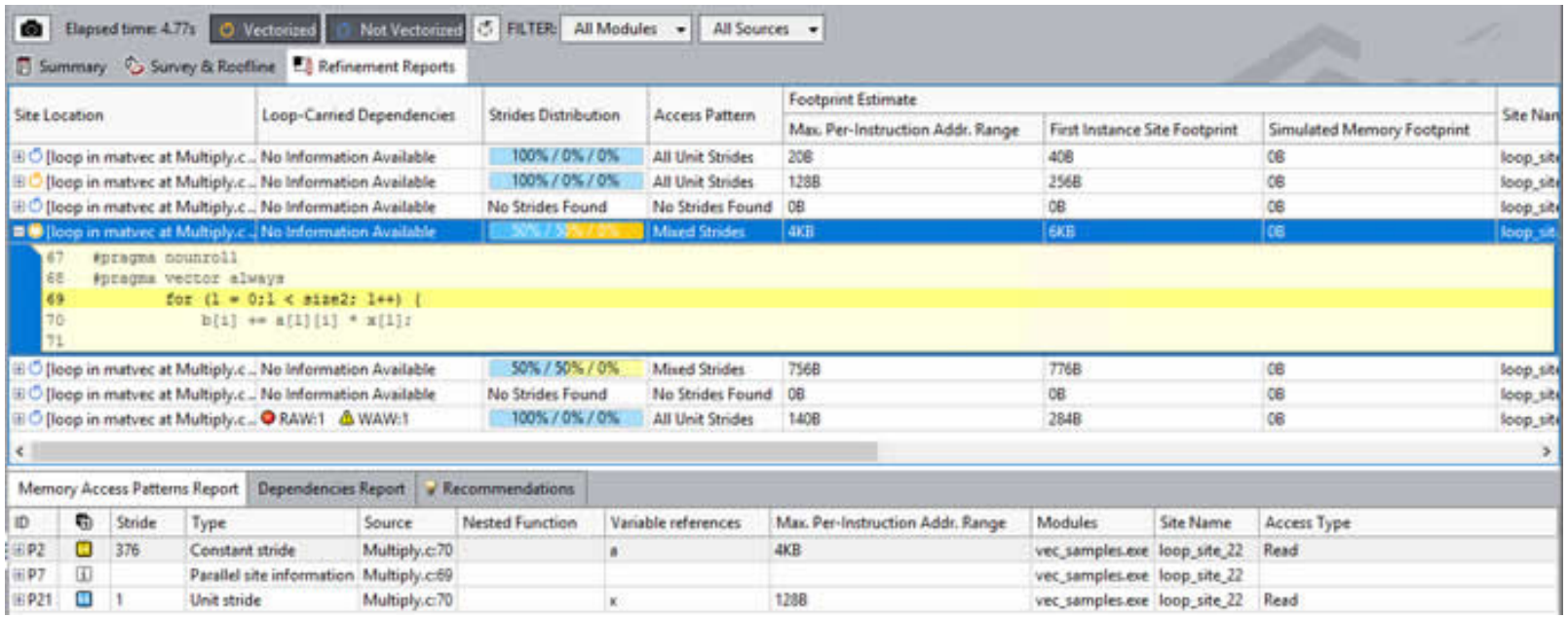
- Permite hacer un análisis de patrones de acceso a memoria: *Memory Access Patterns (MAP) Report*
  - *Uniform*: stride 0
  - *Unit stride* (stride 1)
  - *Constant stride* (stride N)
  - *Irregular stride*
  - *Gather (irregular) stride*

Vectorización y Acceso a memoria

- Patrones de acceso 0 y 1: buenas eficiencias
- Patrones de acceso no consecutivos: pérdida significativa de rendimiento

Intel Advisor (Memory Access)

- Para recolectar la información con la herramienta de *Memory Access Analysis*:
  1. Seleccionar las cajitas de los bucles que se desea hacer el análisis más profundo con 🔥
  2. Pulsar sobre el botón de Collect



Intel Advisor

- En el Ejemplo 4 se ha preparado el **Makefile** para poder usar el Intel Advisor:
  - `make survey`
  - `make open-gui`
  - `make roofline`

gprof

- Herramienta para obtener el perfil de una aplicación.
- El compilador añade llamadas *mcount* para monitorización

Como funciona

1. Compilar el código con la opción -**g**(debug) y -**pg**
2. *mcount* monitoriza las llamadas a cada función y genera un grafo de llamadas
3. *mcount* contabiliza el número de llamadas externas a cada función
4. Después de instrumentalizado el código... se genera el fichero **gmon.out** almost as efficient as normal build
5. Visualiza el contenido con la herramienta **gprof**

gprof

Terminal #1

```
user@lab:~ $ icc -o test_program test.c -g -pg
user@lab:~ $ ./test_program
user@lab:~ $ gprof ./test_program
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative self              self   total
time seconds    seconds calls s/call s/call name
33.86 15.52     15.52    1    15.52  15.52  func2
33.82 31.02     15.50    1    15.50  15.50  new_func1
33.29 46.27     15.26    1    15.26  30.75  func1
0.07  46.30     0.03                       main
...
index % time self children called name

[1]   100.0  0.03  46.27          main [1]
             15.26 15.50    1/1      func1 [2]
             15.52 0.00     1/1      func2 [3]
-----------------------------------------------
             15.26 15.50    1/1      main [1]
[2]   66.4   15.26 15.50    1     func1 [2]
             15.50 0.00     1/1      new_func1 [4]
-----------------------------------------------
             15.52 0.00     1/1      main [1]
[3]   33.5   15.52 0.00     1     func2 [3]
-----------------------------------------------
             15.50 0.00     1/1      func1 [2]
[4] 33.5     15.50 0.00     1     new_func1 [4]
-----------------------------------------------
...
```

valgrind

- Perfilador de memoria: *memcheck*
- Perfilador de carreras: *helgrind* (problemas de sincronización entre hilos)

valgrind

- Herramienta *memcheck* para *memory leaks*

valgrind_hello_good.c

```
#include <stdlib.h>
int main()
{
  return 0;
}
```

Terminal #1

```
user@lab:~ $ gcc -Wall -gstabs valgrind_hello_good.c -o valgrind_hello_good
user@lab:~ $ valgrind --tool=memcheck  --leak-check=full -v ./valgrind_hello_good

==17624== Memcheck, a memory error detector
==17624== Copyright (C) 2002-2011, and GNU GPLd, by Julian Seward et al.
==17624== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==17624== Command: ./valgrind_hello_good

==17635== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==17635== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

valgrind

- Herramienta *memcheck* para lecturas/escrituras ilegales, *memory leaks*, variables sin inicializar...

valgrind_hello_bad.c

```
#include <stdlib.h>
int main()
{
  void* ptr=malloc(1);
  return 0;
}
```

Terminal #1

```
user@lab:~ $ gcc -Wall -gstabs valgrind_hello_bad.c -o valgrind_hello_bad
user@lab:~ $ valgrind --tool=memcheck  --leak-check=full -v ./valgrind_hello_bad

==17722== HEAP SUMMARY:
==17722==     in use at exit: 1 bytes in 1 blocks
==17722==   total heap usage: 1 allocs, 0 frees, 1 bytes allocated
==17722==
==17722== Searching for pointers to 1 not-freed blocks
==17722== Checked 55,996 bytes
==17722==
==17722== 1 bytes in 1 blocks are definitely lost in loss record 1 of 1
==17722==    at 0x402BE68: malloc (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==17722==    by 0x80483F8: main (valgrind_hello_bad.c:8)
==17722==
==17722== LEAK SUMMARY:
==17722==    definitely lost: 1 bytes in 1 blocks
==17722==    indirectly lost: 0 bytes in 0 blocks
==17722==      possibly lost: 0 bytes in 0 blocks
==17722==    still reachable: 0 bytes in 0 blocks
==17722==         suppressed: 0 bytes in 0 blocks
==17722==
==17722== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
==17722== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```