Práctica 1. Vectorización, paralelismo de datos Computación de Altas Prestaciones

Carlos García Sánchez

22 de septiembre de 2021

- "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

- 1 Introducción
- 2 Práctica
- 3 Intel Advisor
- 4 Otras herramientas útiles



Equipo del laboratorio

- Modelo del procesador
- Número de cores
- Vectorización: sse, avx...
- Especificaciones a consultar en la página de Intel¹

```
Terminal #1

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



Equipo del laboratorio

Compiladores

- GNU-GCC: gcc https://gcc.gnu.org
- Intel Compiler: icc 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 .bashro

    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)

Scalar Instructions

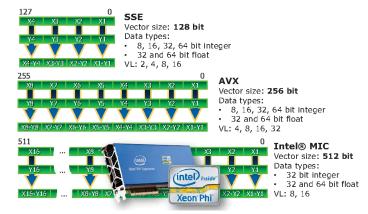
$$4+1=5$$
 $0+3=3$
 $-2+8=6$
 $9+7=2$

Vector Instructions



Soporte operaciones vectoriales

Introducción en el ISA de Intel a partir de 1998





¿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];</pre>
```

Vectorización explícita (intrínsecas)

$vector Add_instrisic.c$

```
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);
__m512_store_pd(c, C_v);
```

■ Vectorización guiada (#pragmas)



Opciones de compilación

- Especifica las optimizaciones en compilación²: -O[n]
 - Por defecto el compilador utilizar -O2

00	Deshabilità cualquier optimizacion
01	Habilita optimizaciones que incrementan tamaño código pero lo acelera
	Optimización global: analisis de flujo, análisis de vida de variables
i i	Deshabilita inlining y uso de intrinsecas
02	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, eliminació de cómputo inútil,
	asignación de registros globales, especulación de control (saltos), desenrrollado bucles,
i i	renomabrado variables, optimizaciones de pila
03	Opt. más agresiva: fusión de bucles, unroll&jam
	Adecuada para código basados en bucles y cómputo fp



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

Contenidos

- Ejemplo 1: opciones de optimización
- Ejemplo 2: autovectorización
- Ejemplo 3: optimización con memoria
- Ejemplo 4: cálculo de potencial eléctrico
- Ejemplo 5: nbody
- Ejemplo 6: Black-Scholes
- Ejemplo 7: stencil (intrínsecas)

Extra

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



Ejemplo 1: 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);
}</pre>
```

```
Terminal #1

user@lab:- $ icc -o foo00 foo00.c -00 -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 -00 -qopt-report=3 -xhost -lm
```



```
Terminal #1
user@lab:~ $ icc -o fooD2 fooD2.c -D2 -gopt-report=3 -xhost -lm
user@lab:~ $ more foo@2.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 -02 -qopt-report=3 -xhost -1m
   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 -02 -qopt-report=3 -xhost -1m
LOOP BEGIN at fooD0.c(23.2) inlined into fooD0.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 ---
```



- 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

```
ejemplol.c

void foo( float* sth, float* theta, int p) {
    for(int i = 0; i < p; i++)
        sth[i] = sin(theta[i]+3.1415927);
}</pre>
```

```
Terminal #1

LIGH BBUIN at food).c(22,2) inlined into food).c(58,2)
remark #15500: LODP WAN VENTURIZED
remark #15500: LODP WAN VENTURIZED
remark #15500: unsealed unlighed unit stride loads: 1
remark #15500: unsealed unlighed unit stride loads: 1
remark #15500: unsealed unlighed unit stride loads: 1
remark #15500: — begin weare or out summary ...
remark #15500: — begin weare or out summary ...
remark #15500: — begin weare or out summary ...
remark #15500: — begin weare loads ...
remark #15500: — weare loads ...
remark #15500: wetcomind math library calls: 1
remark #15500: wetcomind math library calls: 1
remark #15500: ... end wector cost summary ...
LODP END
```



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

```
ejemplol.c

void foo( float* sth, float* theta, int p) {
    for(int i = 0; i < p; i++)
        sth[i] = sin(theta[i]+3.1415927f);
}</pre>
```

```
Terminal #1

LODP BEDIN at fewD2_conversion.e(22.2) inlined into fowD2_conversion.e(58.2)

remerk #15000: LODP WAS VENTMEXED:

remerk #15000: union was been executed in remainder

remerk #15000: unashed unaligned unit stride loads: 1

remerk #15000: unashed unaligned unit stride loads: 1

remerk #15000: unashed unaligned unit stride stores: 1

remerk #15000: consider cost: 1

remerk #15000: occlar cost: 1000

remerk #15000: was remerk #15000

remerk #15000: wetorized entable library colls: 1

remerk #15000: wetorized entable library colls: 1
```



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

```
Terminal #1

1009 BOIN at foot?_conversion.c(23.2) inlined into foot?_conversion.c(58.2)
remark #15500: 1009 VAN VENTONIZED
remark #15500: 1009 VAN VENTONIZED
remark #15500: most value by a seasonated in remainder
remark #15500: most value of the stores: 1

remark #15500: — begin veror root summary ——
remark #15500: — begin veror root summary
remark #15500: — veror root summary
15500
remark #15500: vector root summary
15500
LOOP END
...
```



Ejemplo 3: alineamiento memoria

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

```
cjemplo1_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);
}</pre>
```

```
Terminal #1

LDGF HEDIN at foot2_aligned.c(34,2) inlined into foot2_aligned.c(59,2)
remark #15500.lDGF MS WINTHENIZED

remark #15500.lDGF MS WINTHENIZED

LDGF MS WINTHINIZED

LD
```



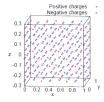
- Consideramos a distribución de un conjunto de m partículas posicionados en $\overrightarrow{r} = (r_{i,x}, r_{i,y}, r_{i,z})$ y con cargas q_i
- Si queremos calcular el potencial eléctrico $\phi\left(\overrightarrow{R}\right)$ en múltiples localizaciones $\overrightarrow{R}_i = (R_{i,x}, R_{i,y}, R_{i,z})$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

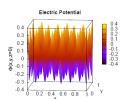
coulomb.c

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





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



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].g / sqrt(dx*dx+dv*dv+dz*dz); // Coulomb's law
```



■ Generando el informe: make REPORT=yes

```
Terminal #1

user@lab:- $ make REPORT=yes
user@lab:- $ icpc -g -std=c++11 -02 -I. -qopt-report=5 -c -o main.o main.cc
icpc: remark #10397: optimization reports are generated in *.optrpt files in the output location
```

- 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 -02 -I. -qopt-report=5 -qopt-report-phase=vec -qopt-report-
filter="main.cc,15-25" -c -o main.oc
```



```
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
```



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

```
Terminal #1
user@lab:~ $ ./coulomb
Initialization... complete.
        Time, s GFLOP/s
    1_4,497e+00
    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
Average performance:
                               2.7 +- 0.2 GFLOP/s
* - warm-up, not included in average
```



- 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 #15405: unmasked strided loads: 4
remark #15475: --- begin vector cost summary ---
remark #15476: scalar cost: 82
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
```



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



 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 * v; // ...y-coordinates...
   float * z; // ...etc.
   float * q; // Charges
   int * type; // Positive or negative charge
1:
```







- 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
        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
```

- **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







- 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



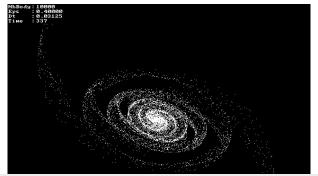
■ En el Heap: el array puede ser reservado/liberado con funciones malloc/free especiales:

memoryAligned.c

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

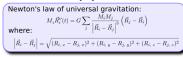


- En astrofísica se usa para resolver problema de la predicción de los movimientos individuales de un grupo de objetos celestes que interactúan entre sí gravitacionalmente
 - Ej: sistemas tierra-luna-sol





Gravitational N-body dynamics:







Cálculo de fuerzas gravitatorias

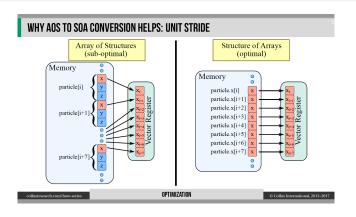
$$\begin{aligned} F_{i,j} &= m_i \frac{\partial^2 R_i}{\partial^2 t} \\ F_{i,j} &= \frac{G m_i m_j (R_j - R_i)}{\left\| R_j - R_i \right\|^3} \\ m_i \frac{\partial^2 R_i}{\partial^2 t} &= \sum_{j=0, i \neq j}^{N} \frac{G m_i m_j (R_j - R_i)}{\left\| R_i - R_i \right\|^3} = \frac{\partial U}{\partial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



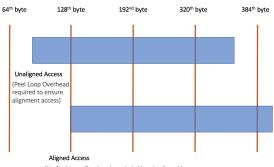
Optimizaciones a estudiar

- 5.1 versión original
- 5.2 vectorización autovectorizada
- 5.3 vectorización guiada
 - Eliminación de "dependencias" en bucles (desalineación de punteros o #pragma)
 - Eliminación de conversión de tipos de datos (float/double)
- 5.4 Accesos a memoria (¿stride?): SoA vs AoS
- 5.5 Memoria alineada









(No Peel Loop Overhead needed. Already aligned.)



Ejemplo 6 (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



Ejemplo 6 (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



Ejemplo 6 (Black-Scholes)

Optimizaciones a estudiar

- 6.1 versión original
- 6.2 vectorización guiada (dependencias)
- 6.3 Memoria alineada



Ejemplo 7 (intrinsecas)

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

Operador stencil

■ Tratamiento de imágenes



Ejemplo 7 (intrínsecas)



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





Ejemplo 7 (intrínsecas)

```
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);
```



Ejemplo 7 (intrínsecas)

Optimizaciones a estudiar

- 7.1 versión original
- 7.2 vectorización guiada (dependencias) + memoria alineada
- 7.3 uso de intrínsecas ³
 - Aritmética en entero (8 bits) con saturación
 - Add/sub packed unsigned 8-bit integers using saturation

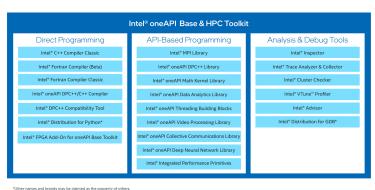
https://software.intel.com/sites/landingpage/IntrinsicsGuide/



³The Intel® Intrinsics Guide:

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





^{*}Other names and brands may be claimed as the property of other



- 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 Linux⁴



⁴Intel Advisor https://drive.google.com/drive/u/0/folders/ 1Jw-nb3XmmoR01zVVcisWNFtnWgDpHFBs



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



⁵Intel Advisor:

https://software.intel.com/en-us/get-started-with-advisor ⁶Video 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

Vectorization Workflow	Threading Workflow
OFF Batch mode	
Run Roofline	
Collect 🖿 🗓	
Enable Roofline with Callstacks	
1. Survey Target	
Collect 🕅 🖿 🗔	



Intel Advisor (survey)

- 1 Botón de control para guardar resultados
- **2** y **3** Filtro de resultados
- 10 Modelo roofline
- 11 Resumen: f funciones vectorizadas, bucles vectorizados,
 - f función escalar y o 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...



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

Intel Advisor (recommendation pannel)





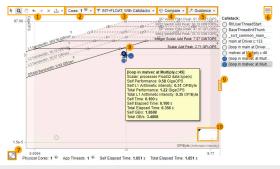
- Permite obtener modelo roofline
- Detectar posibles problemas (Code Analysis)





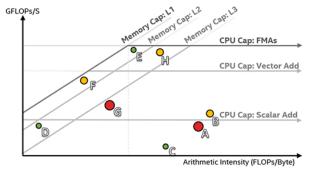
Intel Advisor (roofline)

- 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*)





- 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

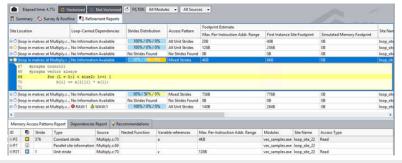
Vectorización y Acceso a memoria

- 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:
 - Seleccionar las cajitas de los bucles que se desea hacer el análisis más profundo con 6
 - 2 Pulsar sobre el botón de la Collect





- 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

- **I** Compilar el código con la opción $-\mathbf{g}(\text{debug})$ y $-\mathbf{pg}$
- 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
- 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
                             15.52 15.52 func2
                             15.50 15.50 new_func1
33.82 31.02
            15.50
33.29 46.27
               15.26
                             15.26 30.75 func1
0 07 46 30
               0.03
                                           main
index % time self children called name
                                 main [1]
     100.0 0.03 46.27
            15.26 15.50
                                    func1 [2]
            15.52 0.00
                                   func2 [3]
                                   main [1]
            15.26 15.50
     66.4
            15.26 15.50
                                 func1 [2]
            15.50 0.00
                                   new func1 [4]
            15.52 0.00
                                    main [1]
                                 func2 [3]
     33.5 15.52 0.00
            15.50 0.00
                                    func1 [2]
[4] 33.5
            15.50 0.00
                                 new func1 [4]
```



valgrind

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



■ 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 LibbEX; 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 ();
    voide ptr=malloc(1);
    return 0;
}
```

```
Terminal #1
user@lab:- $ gcc -Wall -gstabs valgrind_hello_bad.c -o valgrind_hello_bad
user@lab:- $ valgrind --tool=mencheck --leak-check=full -v ./valgrind hello bad
==17722== HEAP SUMMARY:
               in use at exit: 1 bytes in 1 blocks
==17722== total heap usage: 1 allocs, 0 frees, 1 bytes allocated
==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 0x4028E68: malloc (in /usr/lib/valgrind/vgpreload_mencheck-x86-linux.so)
             by 0x80483F8: main (valgrind_hello_bad.c:8)
==17722== LEAK SUMMARY
             definitely lost: 1 bytes in 1 blocks
             indirectly lost: 0 bytes in 0 blocks
             possibly lost: 0 bytes in 0 blocks
             still reachable: 0 bytes in 0 blocks
                   suppressed: 0 bytes in 0 blocks
==17722== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
==17722== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

