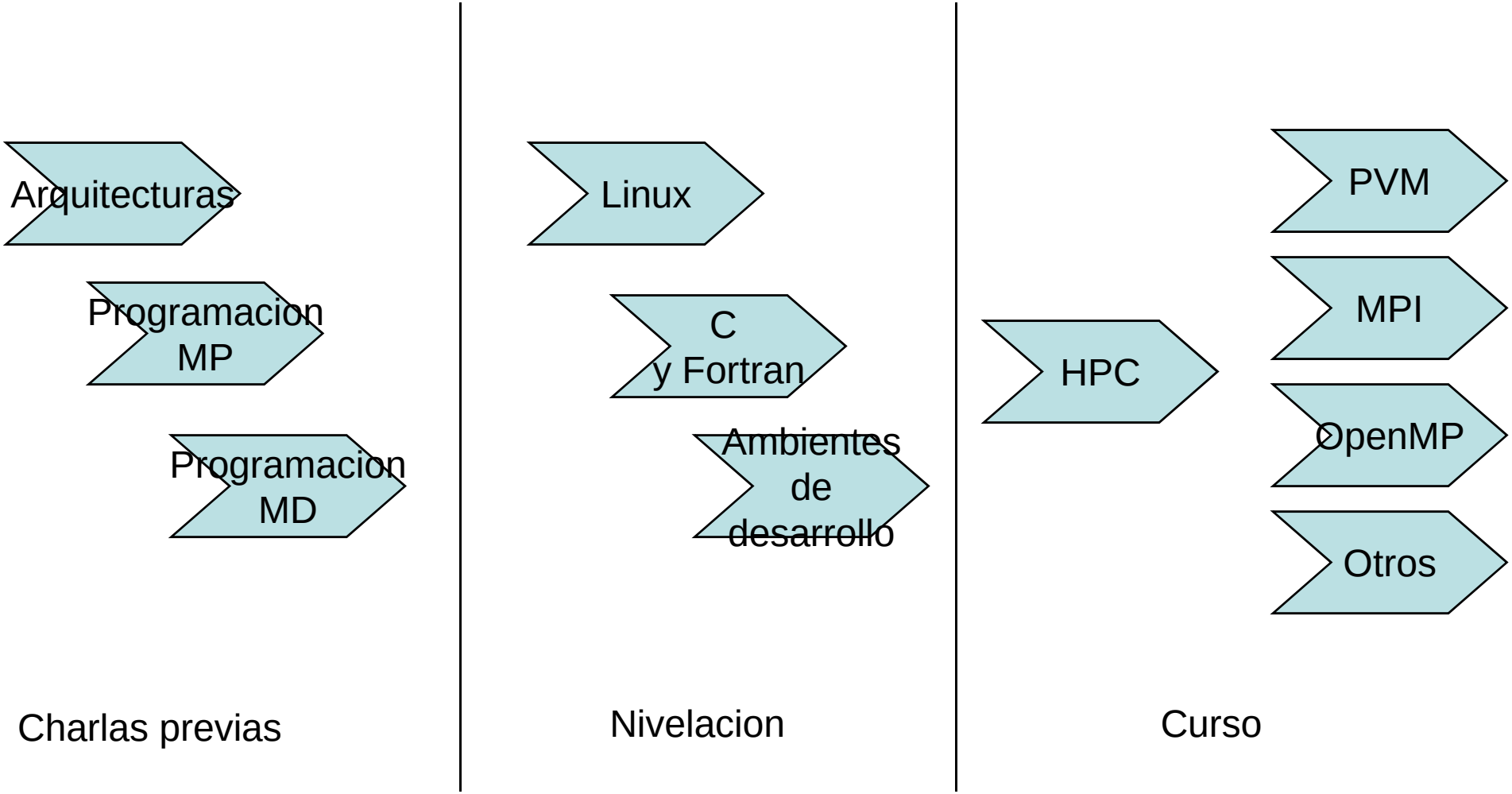
Actualizacion y nivelacion de conocimientos de computación

orientado para futuros participantes del curso de HPC

Julio 2009

Contexto


Arquitecturas
Programacion MP
Programacion MD
Linux
C y Fortran
Ambientes de desarrollo
HPC
PVM
MPI
OpenMP
Otros
Charlas previas
Nivelacion
Curso

Preguntas ?

Agenda

- **DIA 1 (8 de julio): Ambiente LINUX**
  Bibliografia: el entorno de programacion UNIX, de Kernighan y Pike
- **DIA 2 (15 de julio): Los lenguajes de programacion C y FORTRAN**
  Bibliografia: el lenguaje de programacion C, de Kernighan y Ritchie
- **DIA 3 (22 de julio): Ambiente de desarrollo C en LINUX**

Agenda

DIA 3 (22 de julio): Ambiente de desarrollo C en LINUX

- herramientas de desarrollo (gcc, makefiles)

- herramientas de debugging (gdb)

- herramientas de optimizacion (compiladores, profilers)

herramientas de desarrollo

- **Lenguajes interpretados**
  - **PERL, PHP, PYTHON**
- **Lenguajes compilados**
  - **C, C++, FORTRAN**
- **Lenguajes intermedios**
  - **Pseudo-compilados, maquinas virtuales, frameworks**

MATLAB, que tipo de lenguaje es ?

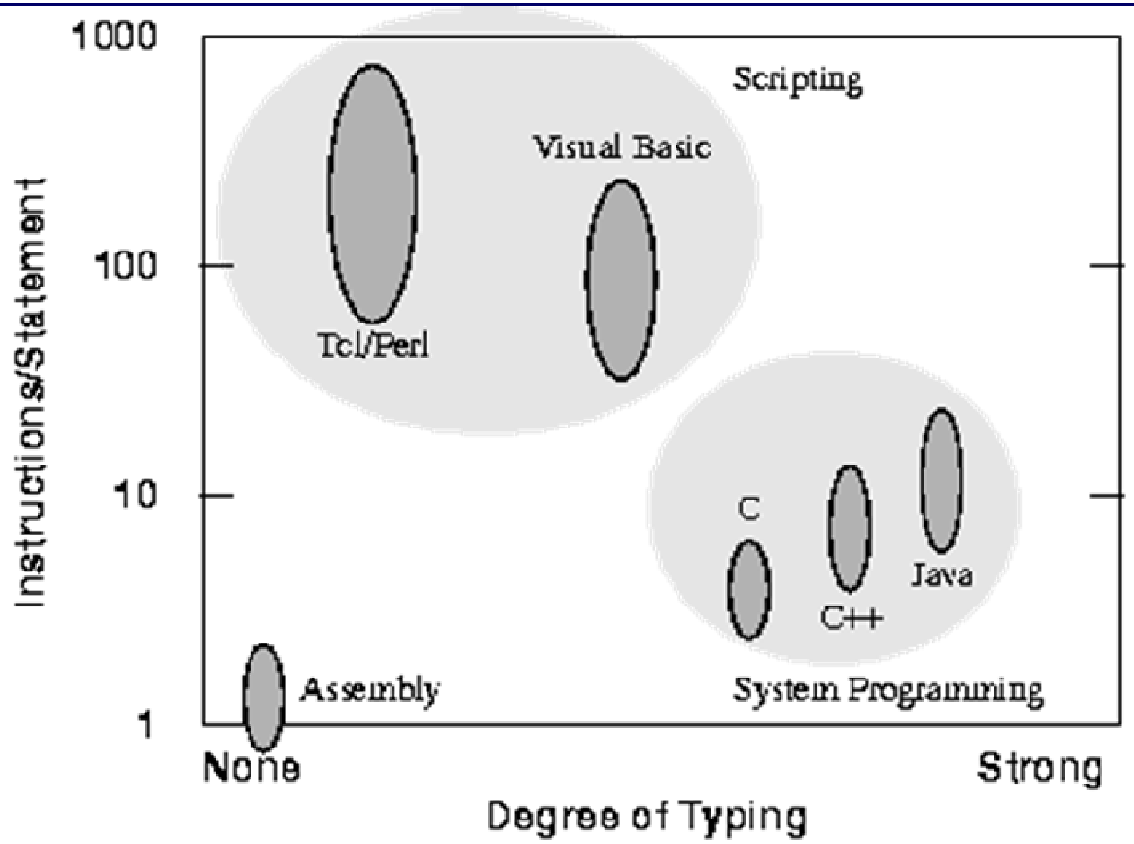
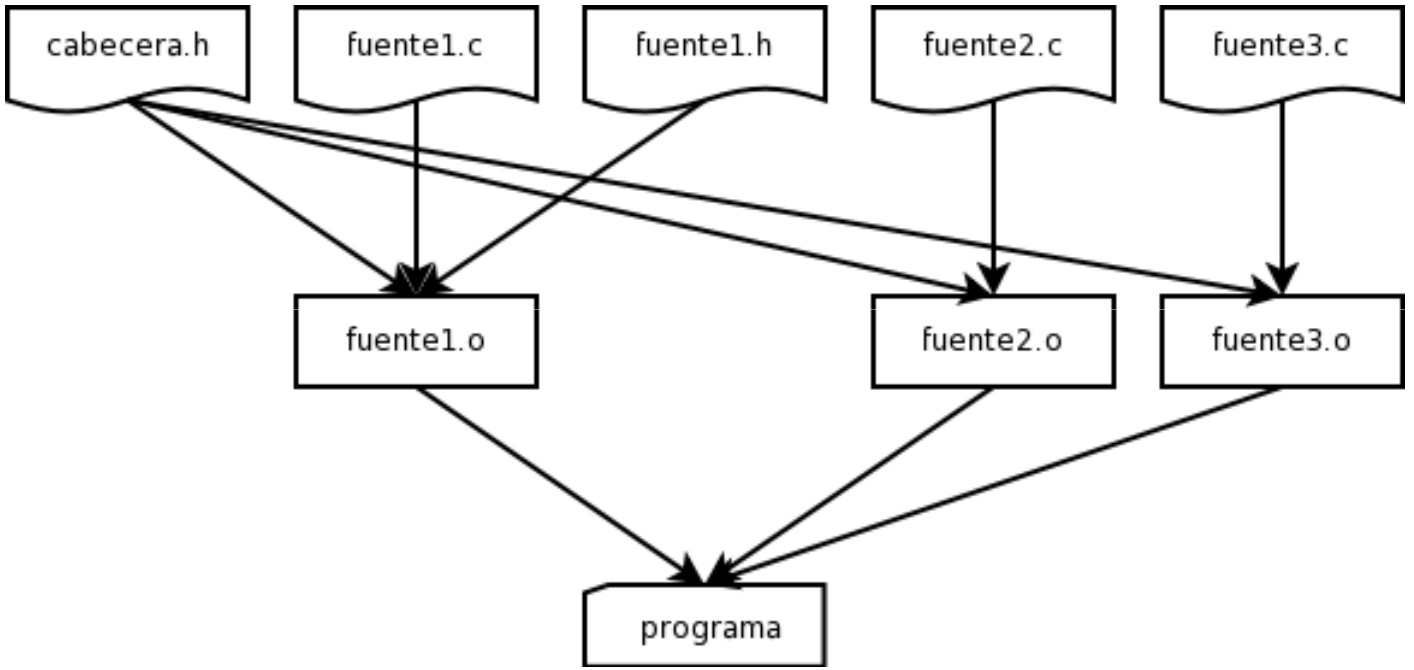


Figure 1. A comparison of various programming languages based on their level (higher level languages execute more machine instructions for each language statement) and their degree of typing. System programming languages like C tend to be strongly typed and medium level (5-10 instructions/statement). Scripting languages like Tcl tend to be weakly typed and very high level (100-1000 instructions/statement).

herramientas de desarrollo



Compilacion

Linkeditacion

herramientas de desarrollo

- GCC = GNU C Compiler
- Traduce codigo fuente a codigo de maquina

```
gcc -o programa fuente1.c fuente2.c fuente3.c
```

```
$ gcc -c fuente1.c #crea fuente1.o
$ gcc -c fuente2.c #crea fuente2.o
$ gcc -c fuente3.c #crea fuente3.o
$ gcc -o programa fuente1.o fuente2.o fuente3.o #crea programa
    ejecutable
```

herramientas de desarrollo

- **Linker**
  - **Permite agregar al programa ejecutable bibliotecas preexistentes (propias o ajenas)**
  - **Tienen que haber sido incluidas en *.h**

  - **Opcion:**

    -lXXX incorpora la biblioteca XXX.a

  - **Ejemplo:**

    -lm incorpora la biblioteca libm.a (matematica)

    gcc -o programa –lm fuente1.c fuente2.c fuente3.c

herramientas de desarrollo

- **Herramienta make**
  - **Automatiza tareas de compilacion**
  - **Interpreta un archivo de definicion de dependencias (makefile)**
  - **En funcion de las fechas de ultima modificacion de los archivos, decide que hacer**

  - **Reglas:**

    Objetivo: dependencia

    <TAB> regla 1

    <TAB> regla 2

herramientas de desarrollo

```
# Makefile de arbol

arbol: arbol.o pinta.o
  gcc –W -Wall -o arbol arbol.o pinta.o

arbol.o: arbol.c pìnta.h
  gcc –W -Wall –c arbol.c

pinta.o: pinta.c pinta.h
  gcc –W -Wall -c pinta.c

clean:
  rm *.o
```

herramientas de desarrollo

- **Macros en makefiles**
  - **Permite definir variables dentro del makefile**

```
CC = gcc
OPCIONES = -g –W -Wall

arbol: $(OBJETOS)
    $(CC) $(OPCIONES) -o arbol $(OBJETOS)
```

herramientas de desarrollo

- **Reglas implicitas**
  - **Permite generalizar reglas**
    - **dependencia es $<**
    - **objetivo es $@**

```
%.o : %.c
	$(CC) $(CFLAGS) $< -o $@
```

herramientas de debugging

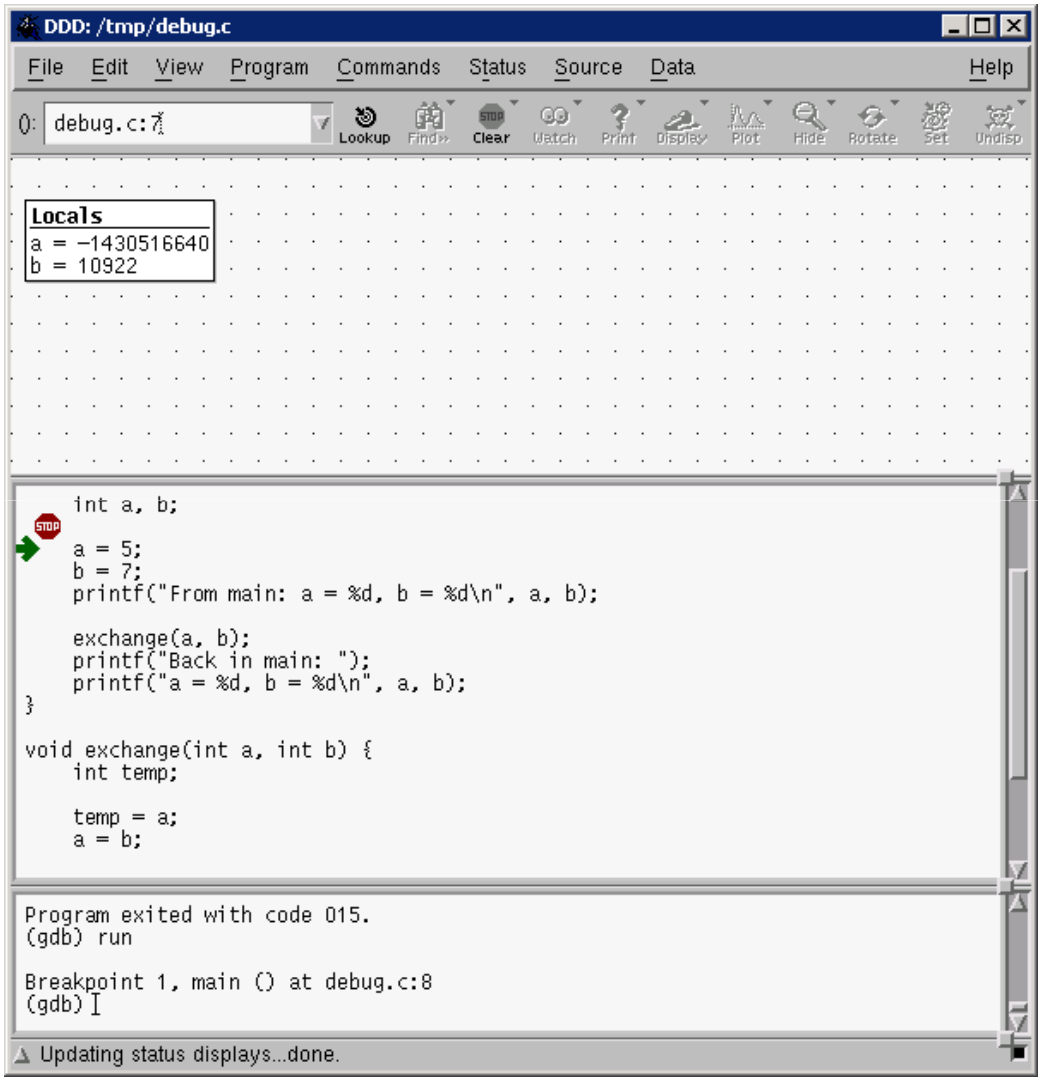
- **Debugger:**
  - **facilita encontrar errores (bugs) en programas**

- **Requieren**
  - **tener el codigo fuente**
  - **haber compilado guardando informacion para debugging (opcion –g)**

herramientas de debugging

```
#include < stdio.h>

void exchange(int a, int b);                                    /* WRONG CODE */

void main() {
   int a, b;

   a = 5;
   b = 7;
   printf("From main: a = %d, b = %d\n", a, b);

   exchange(a, b);
   printf("Back in main: ");
   printf("a = %d, b = %d\n", a, b);
}

void exchange(int a, int b) {
   int temp;

   temp = a;
   a = b;
   b = temp;
   printf(" From function exchange: ");
   printf("a = %d, b = %d\n", a, b);
}
```

herramientas de debugging



herramientas de optimización

- El propio compilador
  - Diferentes niveles de optimizacion alteran el codigo sin modificar el comportamiento
  - Costo: tiempo de compilacion y uso de recursos
  - Opciones de compilacion “-O”

herramientas de optimización

- Ejemplos de optimizacion
  - Constant folding
    - x = 45 * 88 -> x = 3960
  - Subexpression elimination

```
x = a*b;
if(b >= 4)    {
    y = a * b;
    z = 0;
} else   {
    z = a * b * 4;
    y = 0;
}
```

herramientas de optimización

- **Ejemplos de optimizacion**
  - **Reduccion de esfuerzo**
    - **x2 es mas dificil de calcular que x*x**
  - **Eliminacion de codigo no ejecutado**

```
If ( i<1 && i>1 ) {
        // este codigo no es necesario generarlo
}
```

herramientas de optimización

-Os, Optimize for size.

-O0 Reduce compilation time and make debugging produce the expected results. This is the default.

-O, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

-O1 Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function

-O2 Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to -O, this option increases both compilation time and the performance of the generated code.

-O3 Optimize yet more. -O3 turns on all optimizations specified by -O2 and also turns on the -finline-functions, -funswitch-loops, -fpredictive-commoning, -fgcse-after-reload and -ftree-vectorize options.

herramientas de optimización

Restricciones:

no todos los niveles de optimizacion soportan todas las funcionalidades del lenguaje

- aplicaciones multithread
- debugging

herramientas de optimización

- **Herramienta gprof (GNU Profiler,/usr/bin)**
  - **Muestra informacion de cuantas veces es invocada cada funcion, y cuanto tiempo utilizan**
    - **Ejecutables “instrumentados”**
  - **Opcion de compilacion: -pg**

herramientas de optimización

- **PASO 1: ejecucion normal**
  - **Muestra salida normal del programa**
  - **Genera archivo gmon.out**

- **PASO 2: ejecutar gprof + ejecutable**
  - **Genera reporte en pantalla**

```
$ gprof a.out
Flat profile:
Each sample counts as 0.01 seconds.
 %     cumul.    self              self    total
time  seconds seconds    calls us/call us/call name
68.59    2.14    2.14 62135400    0.03    0.03 step
31.09    3.11    0.97   499999    1.94    6.22 nseq
 0.32    3.12    0.01                          main
```

Siguientes pasos…