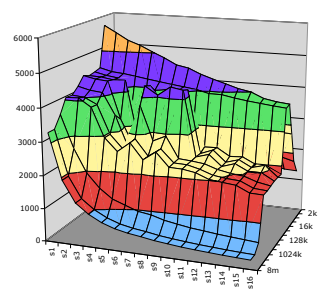
Architecture des systèmes informatiques

I — Plan du cours et Introduction



Georges-Andre.Silber@ensmp.fr
CRI/ENSMP

MINES PARIS
CRI

Thème du cours

- L'abstraction c'est bien
  - Types de données abstraits
  - Analyse asymptotique
- Mais il faut parfois déconstruire
  - Quand il y a des "bugs"
  - Quand on s'intéresse à la performance
- Comprendre comment marche le système !

Fait I
Entiers et réels

`int` n'est pas entier
`float` n'est pas réel

$x^2 \geq 0$?

$(x+y)+z = x+(y+z)$?

```
float : oui !
int :
  40000*40000 = 1600000000
  50000*50000 = ?
```

```
int et uint : oui !
float :
  (1e20+-1e20)+3,14 = 3,14
  1e20+(-1e20+3,14) = ?
```

Arithmétique des ordinateurs

- Pas de valeurs aléatoires
  - Propriétés mathématiques importantes
- Opérations finies
- Propriétés d'anneau sur les `int`
  - Commutatif, distributif, associatif
- Propriétés d'ordre sur les `float`
  - Monotone, valeur du signe

Fait 2

Il faut connaître l'assembleur

Pourquoi l'assembleur ?

- Peu de chances que vous l'utilisiez
  - Les compilateurs sont meilleurs que vous
- Comprendre le modèle d'exécution
  - Comportement des programmes
  - Performance des programmes
  - Logiciels système

Exemple de code

- Compteur de cycles
  - Registre 64 bits spécial (Intel)
  - +1 à chaque cycle
  - Lecture avec `rdtsc`
- Application
  - Mesure du temps (en cycles d'horloge)

```
double t;
unsigned long n;
start_counter ();
P (n);
t = get_counter ();
printf ("P(%ld) required %f clock cycles (t/n=%f).\n", n,t,t/n);
```

Code de lecture du temps

- Code assembleur inclus dans du C (GCC)

  http://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html

```
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;

void access_counter (unsigned *hi, unsigned *lo)
{
  asm (“rdtsc; movl %%edx,%0; movl %%eax,%1”
       : “=r” (*hi), “=r” (*lo)
       :
       : “%edx”, “%eax”);
}

void start_counter ()
{
  access_counter (&cyc_hi, &cyc_lo);
}
```

Code de lecture du temps

```
double get_counter ()
{
  unsigned ncyc_hi, ncyc_lo;
  unsigned hi, lo, borrow;

  access_counter (&ncyc_hi, &ncyc_lo);

  lo = ncyc_lo - cyc_lo;
  borrow = lo > ncyc_lo;
  hi = ncyc_hi - cyc_hi - borrow;
  return (double) hi * (1 << 30) * 4 + lo;
}
```

Mesurer le temps

- Plus compliqué qu'il n'y paraît
  - Nombreuses sources de variation
- Exemple :
  - Somme des entiers de 1 à n

```
unsigned long P(unsigned long n)
{
  unsigned long i;
  unsigned long sum = 1;
  for (i = 2; i < n; i++)
    sum += i;
  return sum;
}
```

Résultats

```
silber@verone:~$ ./compteur 100
P(100) required 1222.000000 clock cycles (t/n=12.220000).
silber@verone:~$ ./compteur 100
P(100) required 1035.000000 clock cycles (t/n=10.350000).
silber@verone:~$ ./compteur 1000
P(1000) required 7335.000000 clock cycles (t/n=7.335000).
silber@verone:~$ ./compteur 1000
P(1000) required 8910.000000 clock cycles (t/n=8.910000).
silber@verone:~$ ./compteur 100000
P(100000) required 702810.000000 clock cycles (t/n=7.028100).
silber@verone:~$ ./compteur 100000
P(100000) required 700342.000000 clock cycles (t/n=7.003420).
silber@verone:~$ ./compteur 10000000
P(10000000) required 72297975.000000 clock cycles (t/n=7.229798).
silber@verone:~$ ./compteur 10000000
P(10000000) required 73844647.000000 clock cycles (t/n=7.384465).
silber@verone:~$ ./compteur 1000000000
P(1000000000) required 7159377240.000000 clock cycles (t/n=7.159377).
silber@verone:~$ ./compteur 1000000000
P(1000000000) required 7190984070.000000 clock cycles (t/n=7.190984).
```

Fait 3

La mémoire a une grande influence

La mémoire

- Est bornée
  - Elle doit être allouée et gérée
  - Beaucoup d'applications sont dominées par elle
- Les bugs de référence mémoire sont pernicieux
  - Effets distants en temps et en espace
- Performance non uniforme
  - Caches et mémoire virtuelle
  - Adaptation des programmes nécessaire

Exemple de bug mémoire

```
f ()
{
  double d = 3.14;
  long int a[2];
  a[2] = 1073741824; /* Hors des bornes */
  printf ("d = %.15g\n", d);
}
```

PowerPC G5 / MacOS X / GCC

```
d = 3.14
```

Intel Xeon / Linux / GCC

```
d = 3.1399998664856
```

Intel Xeon / Linux / GCC -O2

```
d = 3.14
Segmentation fault
```

Erreurs mémoire

- C ou C++ ne fournit pas de protection mémoire
  - Bornes de tableaux / pointeurs invalides
  - Mauvaise gestion `malloc` / `free`
- Bugs difficiles à trouver
  - Effets ? Visibles ? Immédiats ?
- Comment ne pas faire d'erreurs mémoire
  - Java, Lisp, ML, ...
  - Comprendre le fonctionnement
  - Utiliser des outils pour détecter les erreurs

Performance mémoire

- Exemple : copie mémoire

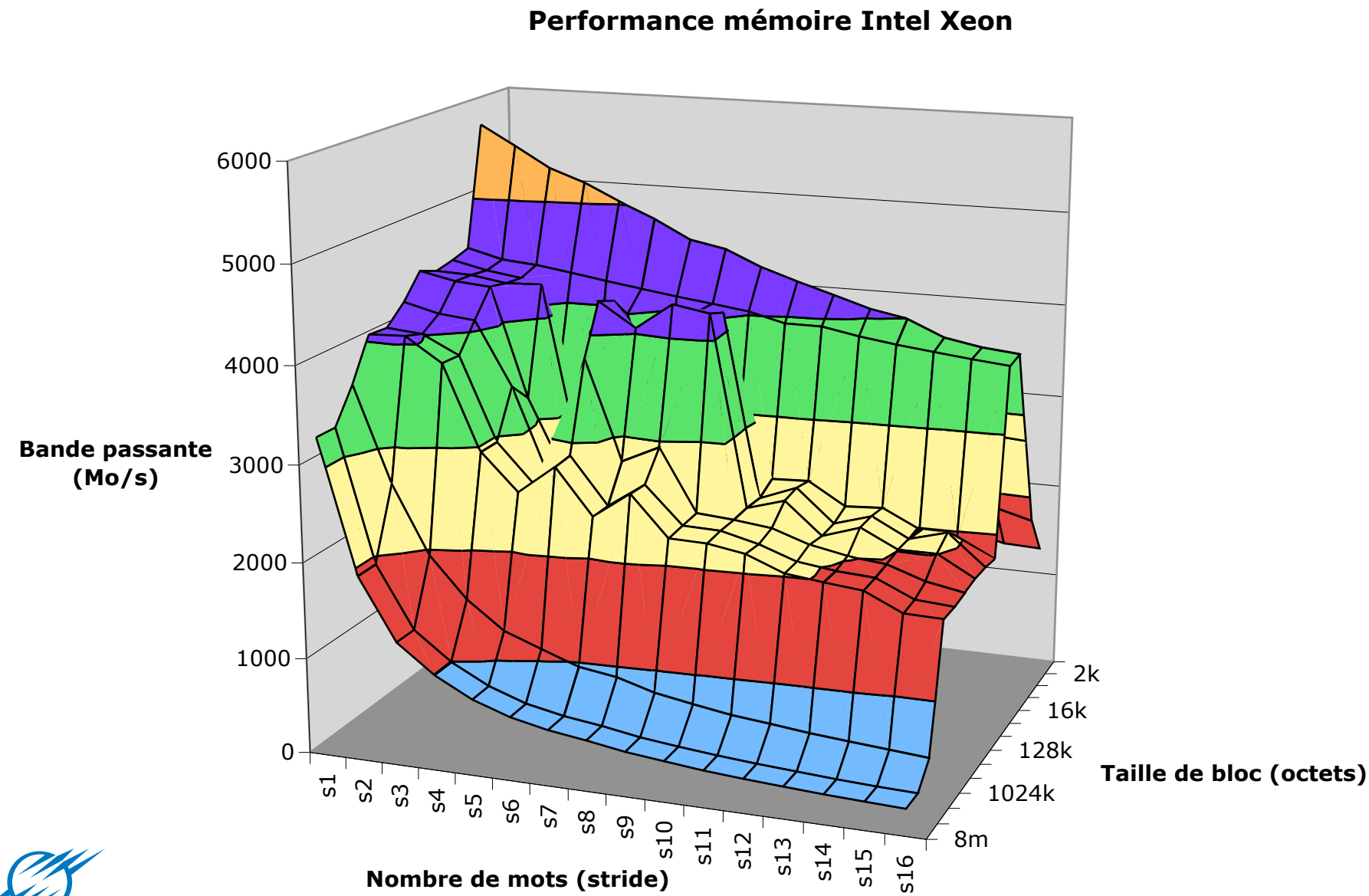
```
void copyij (int src[2048][2048], int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}

void copyji (int src[2048][2048], int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

Intel Xeon / Linux / GCC

```
copyij required 168105788.000000 clock cycles.
copyji required 1614916177.000000 clock cycles.
```

Performance mémoire



Fait 4

Il n'y a pas que la complexité asymptotique

Complexité

- Les facteurs constant comptent !
  - Rapport de 1 à 10 très courant selon le code
  - Optimisation multi-niveaux
- Il faut comprendre le système pour optimiser
  - Comment le programme est compilé et exécuté
  - Comment on mesure les performances
  - Comment on identifie les goulots d'étranglement
  - Comment on améliore le code sans le rendre "illisible" : modularité, généricité, etc...

Fait 5

Un système informatique ne fait pas qu'exécuter des programmes

Entrées / sorties

- Les ordinateurs doivent traiter des données
  - Systèmes d'entrées/sorties sont critiques
  - Performance, fiabilité
- Ils communiquent entre eux via un réseau
  - Grande influence au niveau du système
  - Processus concurrents / parallélisme
  - Media non fiables
  - Interopérabilité
  - Complique encore la mesure des performances

Bibliographie