

---

## Optimisation de programmes : Détection des goulets d'étranglement CPU et GPU

Daniel Etiemble  
de@lri.fr

---

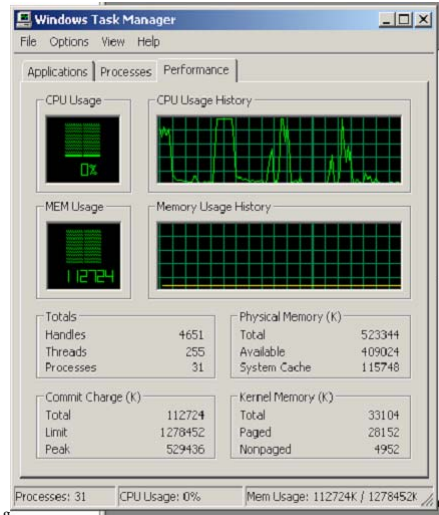
## Evaluation de performance

- Temps d'exécution du programme
- Détection des goulots d'étranglement
  - Détection des fonctions consommatrices de temps
  - Comprendre la cause du goulot d'étranglement
- Outils pour CPU
  - Perfmon
  - Compteurs matériels
  - PROF et GPROF
  - PAPI
  - VTune
- Outils pour GPU
  - NVPerfHUD

## Outils élémentaires d'évaluation de performance

PERFMON  
Gestionnaire de tâches Windows

Information qualitative  
Petit ensemble d'évènements

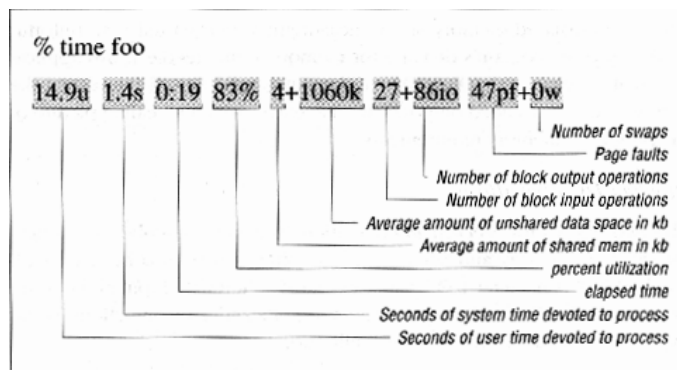


Master Pro  
2005

Optimisations pour g  
D. Etiemble

## Mesurer le temps d'exécution (Linux)

- Commande Time



Master Pro  
2005

Optimisations pour graphique et multimédia  
D. Etiemble

4

## Mesurer le temps d'exécution

---

- <time.h> (Windows)
  - Déclaration
    - clock\_t: begin, end
  - Mesure
    - begin=clock(); // début
      - Temps d'exécution à mesurer
    - end=clock();
    - printf (« temps d'exécution= %f\n", (double) (end-begin));
  - Précis pour des temps d'exécution supérieurs à la ms

## Mesure du temps d'exécution

---

- RDTSC (Read-time stamp counter)
  - Déclaration
    - #define CPUID \_\_asm \_\_emit 0fh \_\_asm \_\_emit 0a2h
    - #define RDTSC \_\_asm \_\_emit 0fh \_\_asm \_\_emit 031h
  - Fonction
    - unsigned \_\_int64 timercall();
  - Mesure

```
temp_cycles1=timercall(); temp_cycles2=timercall();
base= temp_cycles2 - temp_cycles1;
temp_cycles1=timercall();
//CODE A MESURER
temp_cycles2=timercall();
elapsed= temp_cycles2 - temp_cycles1 - base;
printf("te(cycles)= %f\n", (double)(elapsed));
```

## Instruction RDTSC (IA-32 sous Windows)

```
unsigned __int64 timercall(){
    unsigned cycles_low, cycles_high;
    __asm {
        pushad
        CPUID
        RDTSC
        mov     cycles_high, edx
        mov     cycles_low, eax
        popad }

    return ((unsigned __int64)cycles_high << 32)
           | cycles_low;}

```

Mesure: cycles d'horloge

## Instruction RDTSC (IA-32 sous Linux)

```
long long readTSC(void)
{
    long long t;
    asm volatile (".byte 0x0f,0x31" : "=A" (t));
    return t;
}

double dtime()
{
    return (double) readTSC();
}

```

Mesure: cycles d'horloge

## Outils de profilage

---

- PROF (Linux)
  - Temps passé dans chaque fonction ou sous-programme
  - Utilisation
    - `gcc -o file -p file.c` (compile and link with -p option)
    - `file` (run program and produces 'mon.out')
    - `prof file > file.prof`
  - Exemple
- VTune (Windows)

## Sortie Prof

---

Liste de fonctions, triée en ordre décroissant

[index]	secs	%	cum.%	samples	function (dso: file, line)
[1]	23.380	43.0%	43.0%	2338	scatter (montecarlo: montecarlo.c, 266)
[2]	8.410	15.5%	58.5%	841	ran3 (montecarlo: montecarlo.c, 439)
[3]	6.050	11.1%	69.6%	605	__fastm_cos (libfastm.so: cos.s, 73)
[4]	5.730	10.5%	80.2%	573	path (montecarlo: montecarlo.c, 216)
[5]	5.560	10.2%	90.4%	556	__fastm_log (libfastm.so: log.s, 100)
[6]	2.660	4.9%	95.3%	266	getRandomNumber (montecarlo: montecarlo.c, 481)
[7]	2.470	4.5%	99.9%	247	doAnisotropicScatter (montecarlo: montecarlo.c, 119)
[8]	0.040	0.1%	99.9%	4	__fastm_exp (libfastm.so: exp.s, 92)
[9]	0.020	0.0%	100.0%	2	__atan (libm.so: atan.c, 145)
[10]	0.010	0.0%	100.0%	1	__write (libc.so.1: write.s, 20)
	0.010	0.0%	100.0%	1	**OTHER** (includes excluded DSOs, rld, etc.)

## Profilage via graphe d'appels

---

- GPROF (Linux)

- gcc -o file -pg file.c (compile and link with -pg option)
- file ( run program and produces 'gmon.out' )
- gprof file > file.gprof

### Exemple

called/total		parents		called+self called/total	name	index
index	%time	self	descendents			
		3.74	27.14	7/7	.main [1]	
[3]	46.2	3.74	27.14	7	.doAnisotropicScatter [3]	
		9.90	9.20	27621579/27621579	.scatter [4]	
		3.35	4.61	27691579/27691579	.path [9]	
		0.02	0.03	70000/70000	.initPhoton [16]	
		0.03	0.00	70000/70000	.score [18]	
		0.00	0.00	21/53	.printf [55]	
		0.00	0.00	7/7	.printTissueParams [93]	
		0.00	0.00	7/7	.clearIntensityArray [91]	
		0.00	0.00	7/7	.dump1 [92]	

## Compteurs matériels

---

- Pratiquement tout processeur disponible a des compteurs matériels
- Les interfaces et la documentation sont pauvres ou inexistantes
- Le matériel et la sémantique diffèrent selon les CPU
- Utile pour les mesures, l'analyse, l'optimisation, la modélisation et l'évaluation de performance.

## Données qui peuvent être disponibles

---

- Nombre de cycles
- Nb d'instructions flottantes
- Nb d'instructions entières
- Nb de chargements/rangements
- Nb de branchements pris/non pris
- Mauvaises prédictions de branchement
- Suspensions pipeline dues au système mémoire
- Suspensions pipeline dues aux conflits de ressources
- Echecs caches I/D pour différents niveaux
- Invalidations cache
- Echecs TLB
- Invalidation TLB

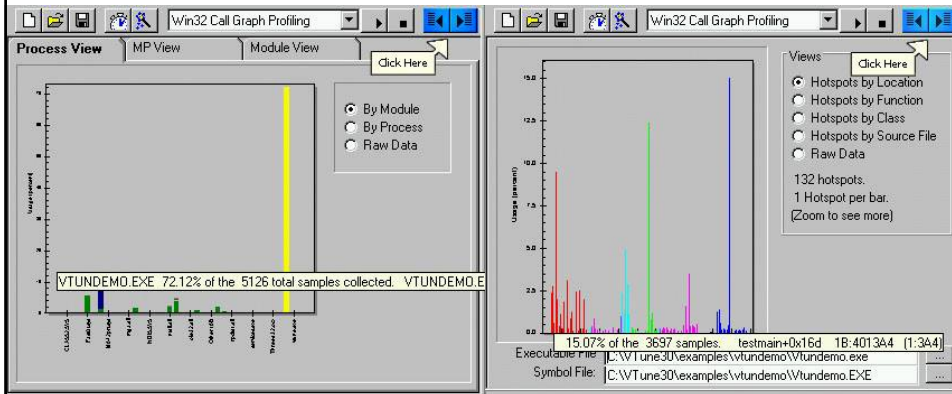
## VTune (Intel)

---

- L'analyseur VTune fournit trois types de techniques d'analyse pour différents aspects de la performance de l'application
  - Analyse basée sur le temps/événements pour exécuter et échantillonner le programme et identifier les goulots d'étranglement dans le code
  - Analyse du code statique pour analyser les fonctions statiques et indiquer les problèmes de performance
  - Profilage du graphe d'appel pour suivre chaque thread et séquence d'appels qui prennent le plus de temps

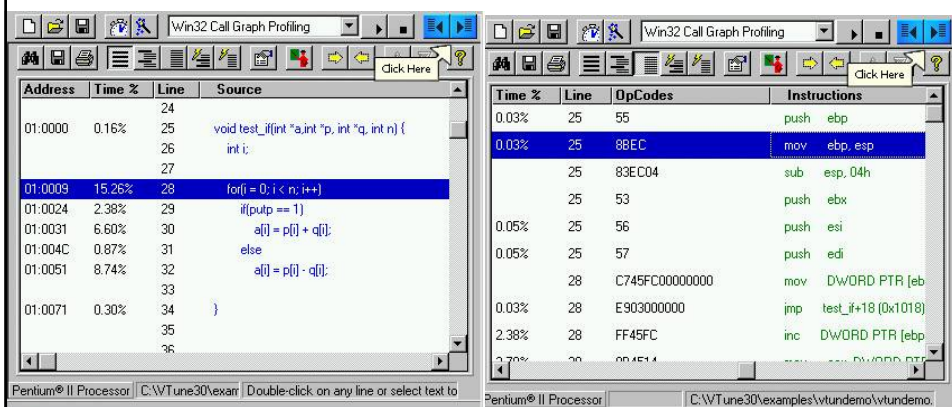
# VTune

## Détection des goulots d'étranglement



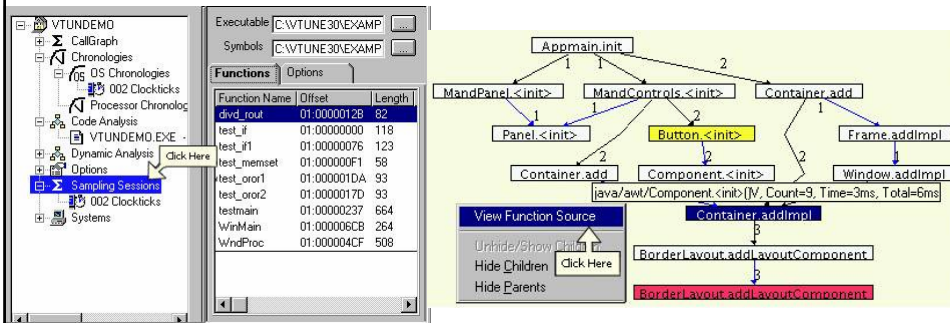
# VTune (Intel)

## Localisation des goulots d'étranglement



# VTune

## Graphe d'appel

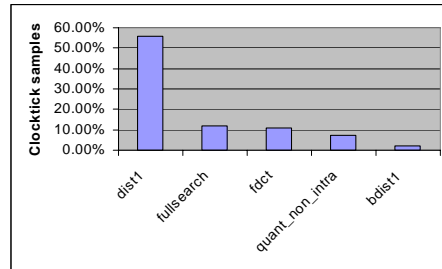


## Amélioration de la performance

- Détection des fonctions critiques
- Comprendre la cause du temps passé dans ces fonctions
  - Mauvais choix entre plusieurs « solutions »
  - Mauvais comportement des caches
  - Possibilité d'améliorer le parallélisme
- Peut-on améliorer la situation ?

## Instructions spéciales (ex : PSABW)

- PSABW
  - Valeur absolue des différences des octets (unsigned char) dans DST et SRC
  - Somme des valeurs absolues pour les huit octets bas dans les 32 bits de la partie basse de DST
  - Somme des valeurs absolues pour les huit octets hauts dans les 32 bits de la partie haute de DST



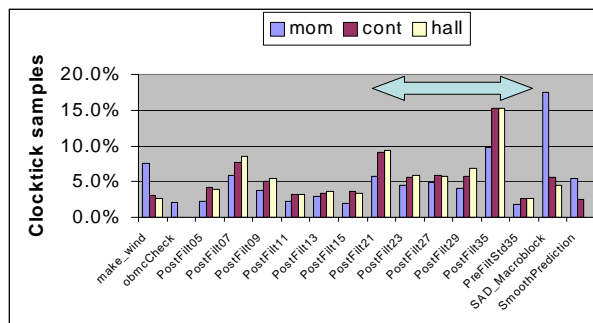
Encodeur MPEG2

$$SAE = \sum_{i=0}^{i=7} \sum_{j=0}^{j=7} |C_{ij} - R_{ij}|$$

0000 RES2 0000 RES1

## A NOUVEAU SUR LE PRODUIT SCALAIRE

### Codec "Matching Pursuit Video" de Berkeley



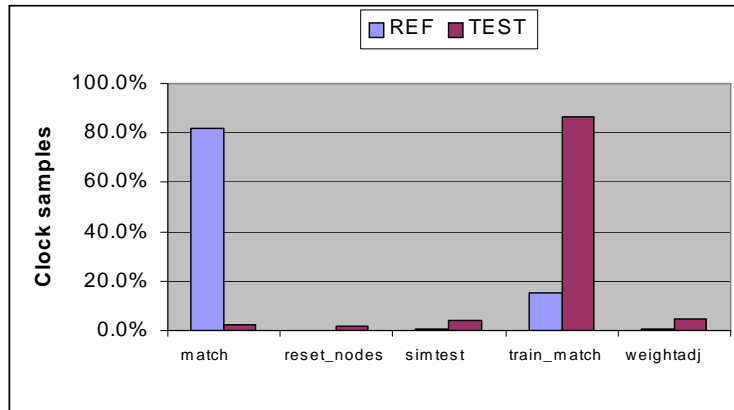
CODEUR

### FONCTIONS CRITIQUES

SAD (DIST1)

Fonctions Post-filtre

## Goulets d'étranglement ART



### Fonctions Match et train match

## Code critique

```
if ( !Y[tj].reset )  
  for (ti=0;ti<numfls;ti++)  
    Y[tj].y += f1_layer[ti].P * bus[ti][tj];
```



*Tableau de structures* ***Pas non unitaire***

- bus[][] et tds[][] interviennent dans différentes fonctions dans le programme
- f\_layer est la structure principale

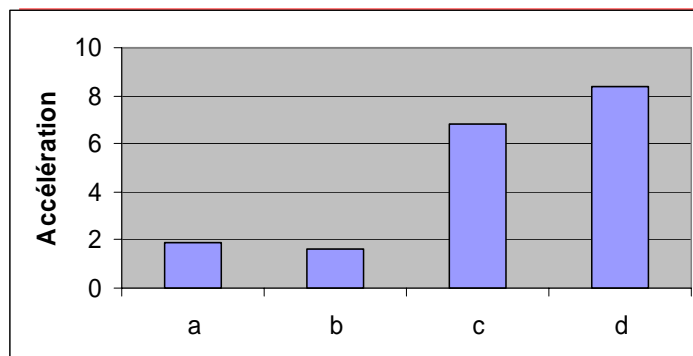
## IMPACT DE LA STRUCTURE

```
typedef struct {  
    double *I;  
    double W;  
    double X;  
    double V;  
    double U;  
    double P;  
    double Q;  
    double R;  
} f1_neuron;  
  
f1_neuron *f1_layer;  
  
for (ti=0;ti<numf1s;ti++)  
    Y[tj].y += f1_layer[ti].P * bus[ti][tj];
```

→ 11 (entrée)  
+ 7  
= 18 doubles par occurrence de la structure

**1 ECHEC/ITERATION !!!!!**  
**10 000 ITERATIONS**

## Optimisation de ART



- a) Remplacer la structure par des tableaux d'éléments
- b) Changer l'ordre des itérations pour Bus[ ][ ] et tds[ ][ ]
- c) Les deux optimisations a) et b)
- d) Les deux optimisations plus division remplacée par multiplication

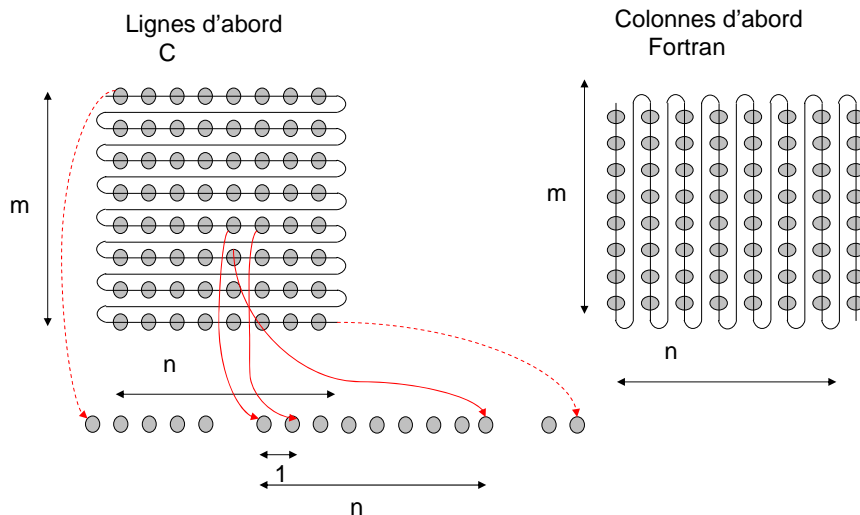
## Division/Multiplication

```
tnorm = sqrt((double) tnorm);  
for (tj=0;tj<numfls;tj++)  
    f1_layerU[tj] = f1_layerV[tj] / tnorm;  
for (tj=0;tj<numfls;tj++)  
    f1_layerX[tj] = f1_layerW[tj] / tnorm;
```

```
itnorm = 1/sqrt((double) tnorm);  
for (tj=0;tj<numfls;tj++)  
    f1_layerU[tj] = f1_layerV[tj] * itnorm;  
for (tj=0;tj<numfls;tj++)  
    f1_layerX[tj] = f1_layerW[tj] * itnorm;
```



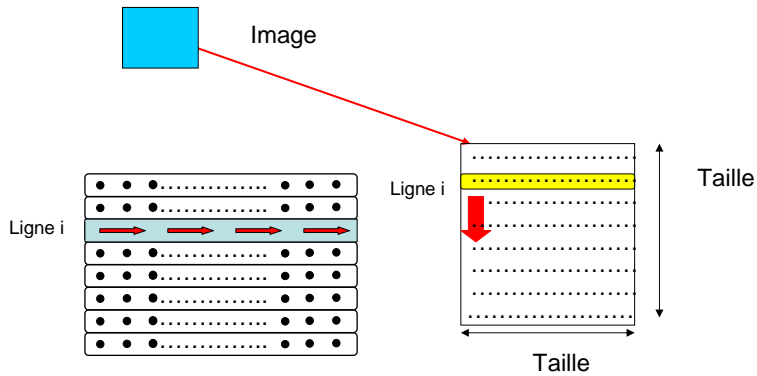
## Placements canoniques



## Ligne d'abord : filtrage horizontal

Ligne d'abord  
Filtrage horizontal

Exploitation de la  
localité spatiale



Master Pro  
2005

Optimisations pour graphique et multimédia  
D. Etiemble

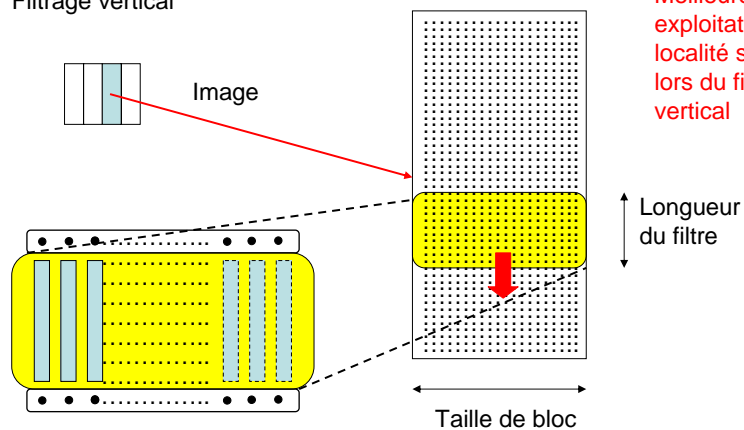
27

## Ligne d'abord : filtrage vertical

Ligne d'abord  
Filtrage vertical

AGGREGATION

Meilleure  
exploitation de la  
localité spatiale  
lors du filtrage  
vertical

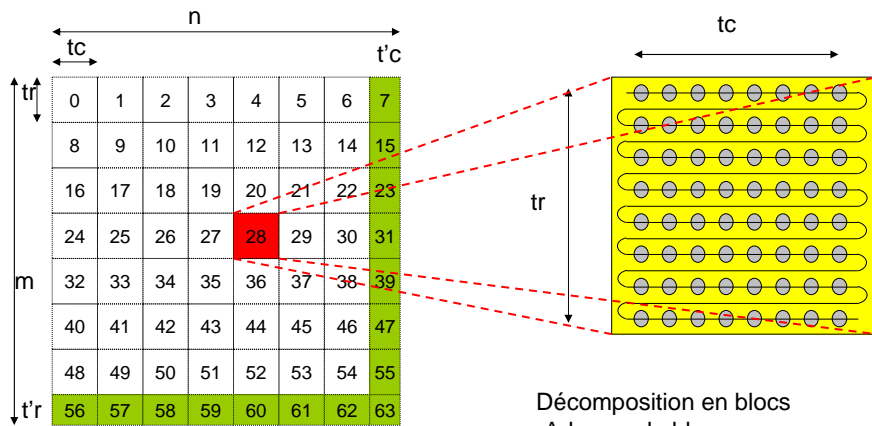


Master Pro  
2005

Optimisations pour graphique et multimédia  
D. Etiemble

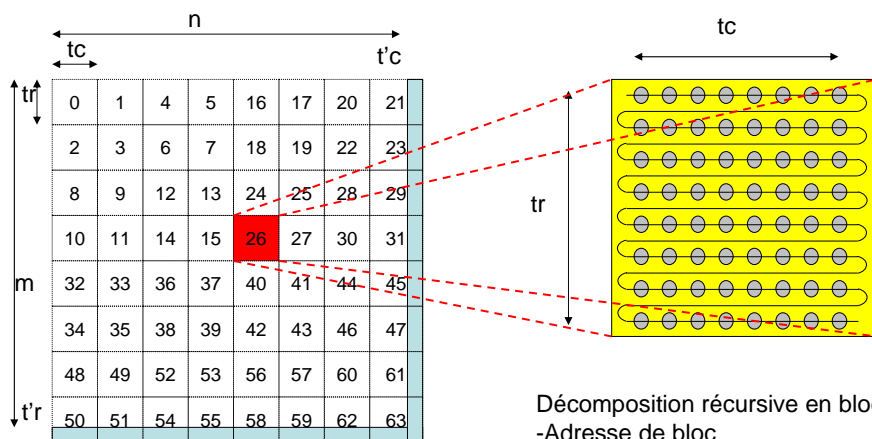
28

## Placement 4D



Décomposition en blocs  
-Adresse de bloc  
-Adresse dans le bloc

## Placement de Morton

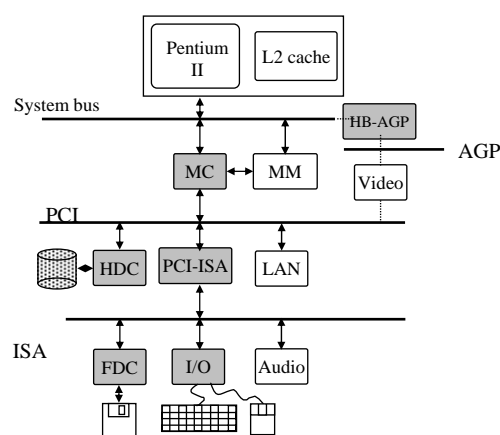


Décomposition récursive en blocs  
-Adresse de bloc  
-Adresse dans le bloc

## Exemple d'utilisation des « bottlenecks »

- L'optimisation des PC «milieu de gamme » chez HP (fin des années 90)
  - PC bas de gamme
  - PC milieu de gamme
  - PC serveurs
- Outils développés pour une division de HP (Grenoble)
  - Cycle de conception court
  - Outils conviviaux
    - faciles d'utilisation
    - Temps d'exécution courts
  - Outils Windows (pas Unix)

## Architecture des PC



- Composants d'étagère
  - Partie Intel
  - Partie HP
- Choix matériel restreint
- Gamme de performance réduite

## Spécificités des PC

---

- Benchmarks pour PC
  - Pas de code source
  - Code Application *et* OS
  - Benchmarks Windows
- Pas de modèles de simulation
  - Processeurs – Chip sets - etc
  - La modélisation est difficile
    - Documentation rare (secrets Intel/AMD)
    - Compromis précision/complexité

## MESURES DE PERFORMANCE



---

- Peu de points d'observation
  - Compteurs de performance du CPU, de Windows
  - Bus
    - Pentium II
    - PCI
    - AGP
- Peu de paramètres peuvent changer
  - Fréquence processeur (ralentir...)
  - Composants
    - Cartes graphiques
    - Disques durs

## PerfTool

---

- Compteurs
  - Semblables à Windows Perfmon
    - S'appuient sur compteurs CPU et Windows
  - Améliorations
    - Les prises de valeur sont synchronisés sur les processus
    - Fichiers de sortie en formats manipulables (Excel, etc)
- Goulets d'étranglement "spécialisés"
  - CPU
  - Mémoire
  - Disques
  - Graphique

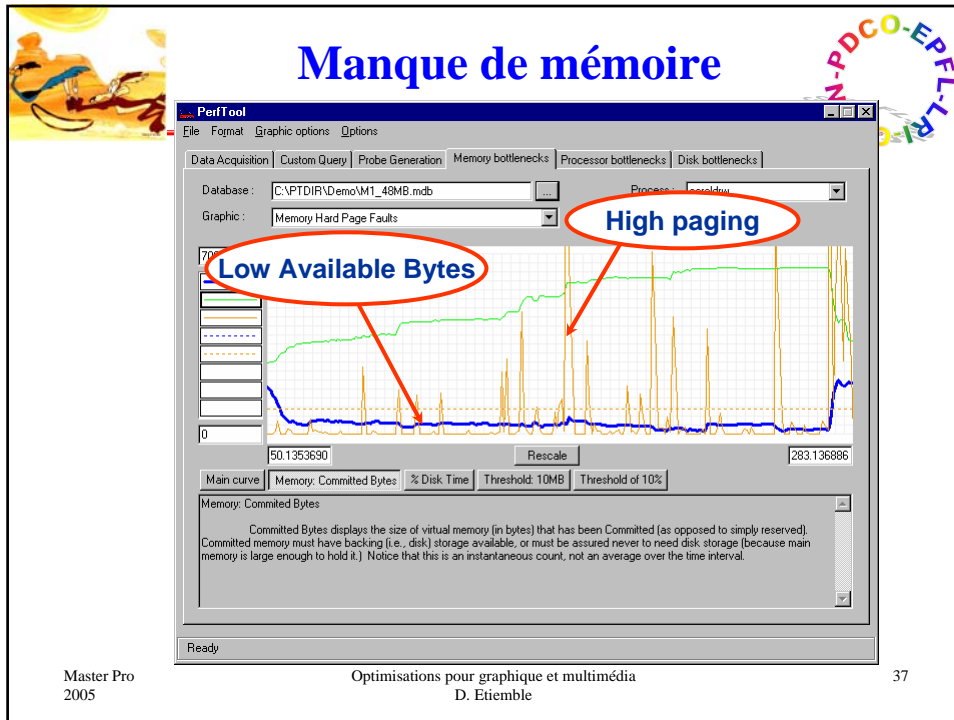


## Bottleneck "mémoire"

---

- Bottleneck mémoire = pas assez de mémoire physique pour la mémoire virtuelle qu'utilisent les processus
  - Le système doit "swapper" des pages disques
  - Beaucoup de défauts de page évitables
- Compteurs à observer
  - Logical Disk (X): % Disk Time
  - Memory: Available Bytes
- => **Bottleneck mémoire =**
  - (Memory: Available Bytes <10MB )**
  - && (Paging > 10% Disk Read Time)**

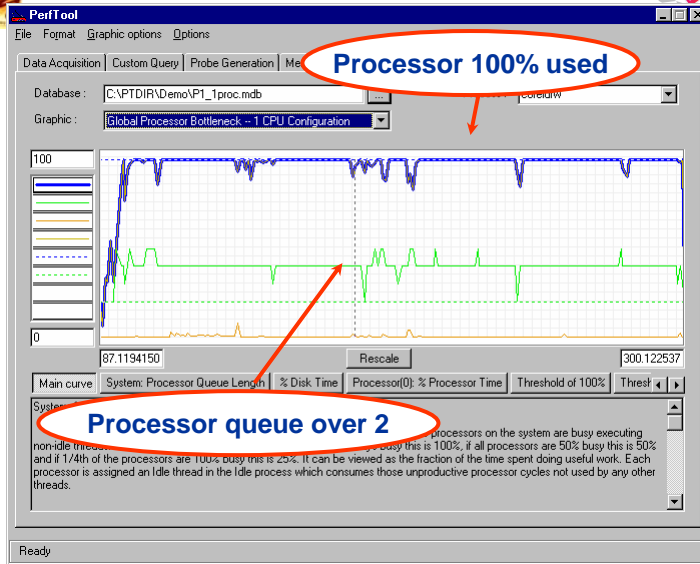
## Manque de mémoire



## Bottleneck processeur

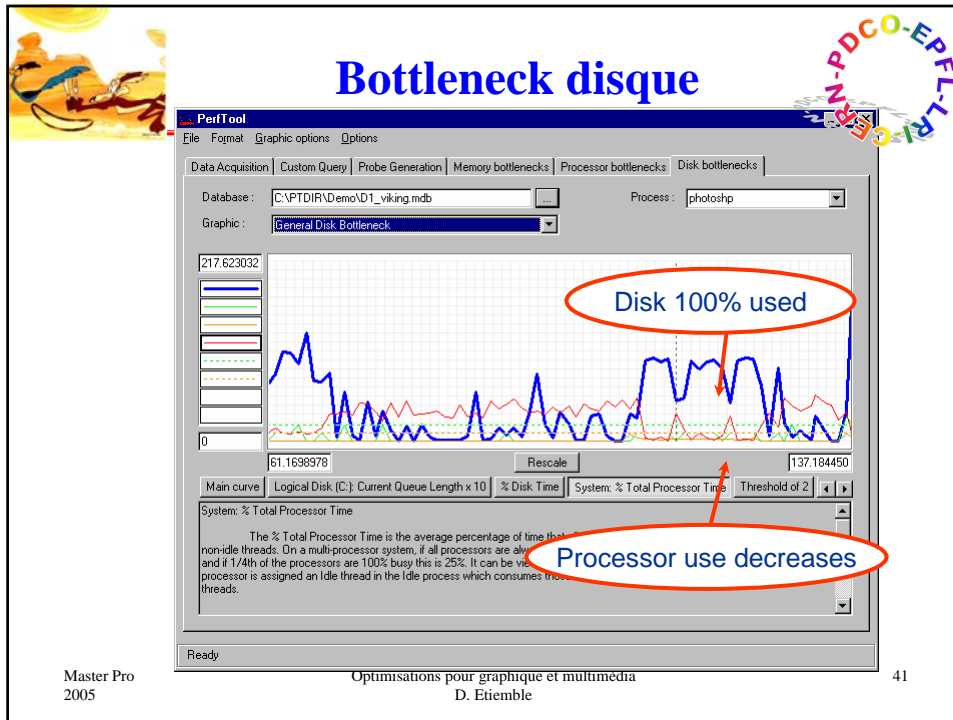
- Processeur trop occupé
  - Files d’attentes longues en permanence
  - Temps de réponse trop important
  - Situation “idéale” : occupation maximale du CPU
- Compteurs à observer
  - System: % Total Processor Time
  - Processor (\*): % Processor Time
  - System: Processor Queue Length
  - **Bottleneck processeur =**  
 (System: Processor Queue Length > 2) && (System: % Total Processor Time > 90%)

## Bottleneck CPU



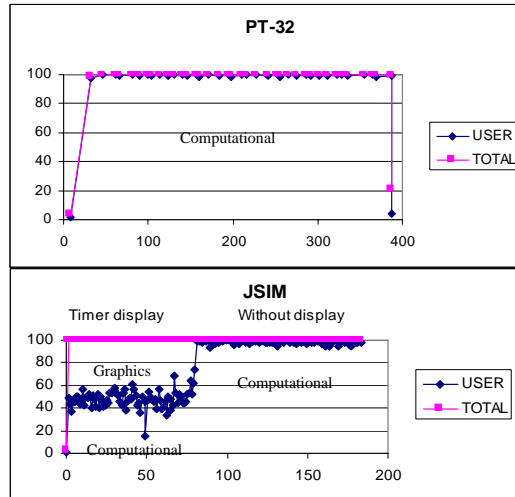
## Bottleneck disques

- Le disque ne peut servir toutes les requêtes suffisamment vite. Le système attend à cause du disque.
- Queues longues en permanence et le processeur est inoccupé
- Compteurs à observer
  - Logical Disk (C:): % Disk Time
  - Logical Disk (C:): Current Disk Queue Length
  - System: % Total Processor Time
  - => **Bottleneck disques =**
    - (Logical Disk (C:): % Disk Time increases to ~100%) && (Logical Disk (C:): Current Disk Queue Length > 2) && (System: % Total Processor Time decreases)



- ## Bottleneck graphique
- 
- Bottlenecks CPU
    - Calcul (CPU)
    - Graphique (CPU+GPU)
  - Activités graphiques avec Windows NT
    - Tâches graphiques en mode “kernel”
  - Différencier “calcul” et “graphique”
    - %CPU user time
    - Quand %CPU user différent de %CPU total =100% alors bottleneck graphique
- Master Pro 2005 Optimisations pour graphique et multimédia  
D. Etiemble 42

## EXEMPLE 1: Calcul ou graphique



CPU32 (Winbench97)

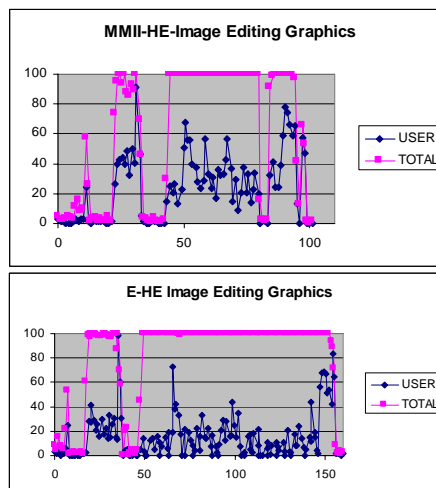
Processor simulator  
- with timer display  
- without timer display

Master Pro  
2005

Optimisations pour graphique et multimédia  
D. Etiemble

43

## EXEMPLE 2: GDI Playback for High-End Graphics



When the graphics board has hardware capabilities, the graphics tasks are executed faster than when the CPU emulates in user mode the graphics tasks. %CPU privileged mode decreases and %CPU user mode increases when the graphics boards have more hardware capabilities.

Matrox Millennium II

ElsaEclipse

Master Pro  
2005

Optimisations pour graphique et multimédia  
D. Etiemble

44

## Architecture du GPU (en 2 mots)

- Architecture pipelinée : chaque unité utilise les données de l'unité précédente pour faire son travail
- Méthode : identifier et éliminer les goulets d'étranglement
- But : équilibrer le pipeline



## Le package NVPerfKit

### Instrumented Driver

GLExpert

NVPerfHUD

NVPerfSDK

NVPerfAPI

Sample Code

Helper Classes

Documentation

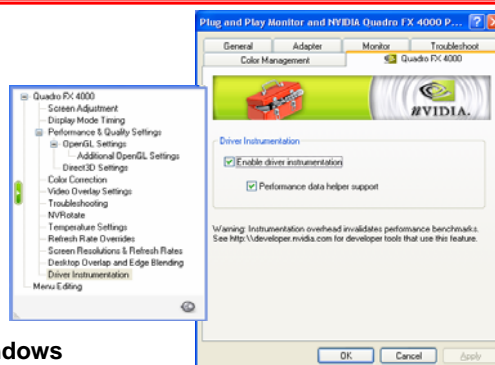
### Tools

NVIDIA Plug-In for

Microsoft PIX for Windows

gDEDebugger

NVDevCPL



## NVPerfKit Instrumented Driver

---

- Exposes GPU and Driver Performance Counters
- Data exported via NVIDIA API and PDH
- Supports OpenGL and Direct3D
- Simplified Experiments (SimExp)
- Collect GPU and driver data, retain performance
  - Track per-frame statistics
  - Gather and collate at end of frame
  - Performance hit 1-2%

## GLExpert: What is it?

---

- Helps eliminate performance issues on the CPU
- OpenGL portion of the Instrumented Driver
  - Output information to console/stdout or debugger
  - Different groups and levels of information detail
- Controlled using tab in NVDevCPL
- What it can do (today)
  - GL Errors: print when they are raised
  - Software Fallbacks: indicate when the driver is in fall back
  - GPU Programs: errors during compile or link
  - VBOs: show where they reside, mapping details
  - FBOs: print reasons a configuration is unsupported
- Feature list to grow with future drivers

## NVPerfKit: Direct3D Counters

---

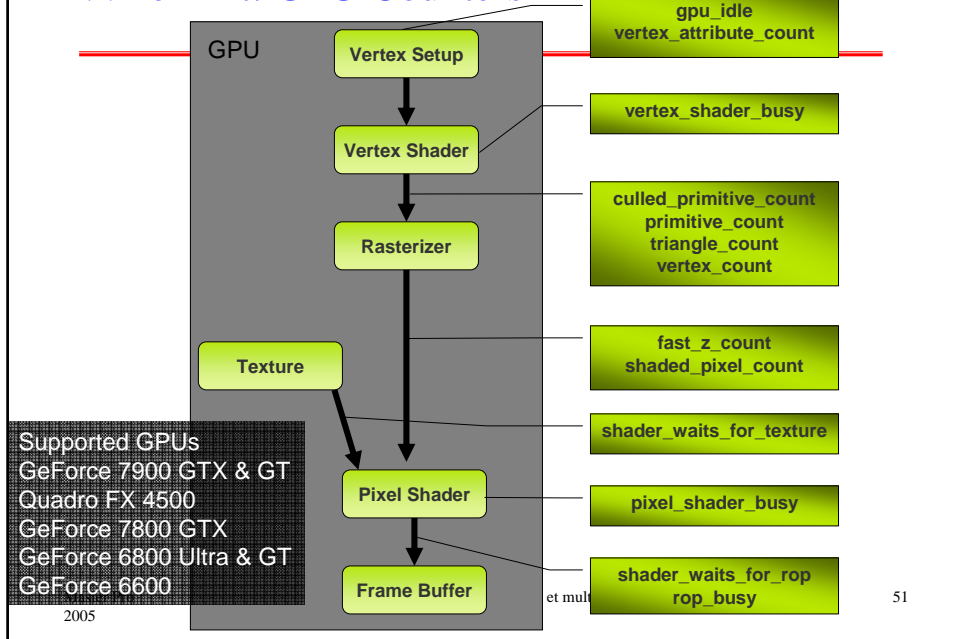
- General
  - FPS
  - ms per frame
- Driver
  - Driver frame time (total time spent in driver)
  - Driver sleep time (waiting for GPU)
- Counts
  - Triangles
  - Instanced triangle
  - Batches
  - Locked render targets
- Memory
  - AGP memory used in MB and bytes
  - Video memory used and total in MB and bytes

## NVPerfKit: OpenGL Counters

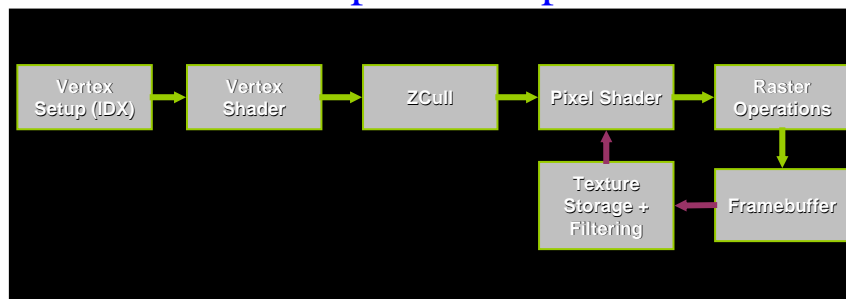
---

- General
  - FPS
  - ms per frame
- Driver
  - Driver frame time (total time spent in driver)
  - Driver sleep time (waiting for GPU)
  - % of the frame time driver is waiting
- Counts
  - Batches
  - Vertices
  - Primitives
- Memory
  - AGP memory used in MB and bytes
  - Video memory used and total in MB and bytes

## NVPerfKit: GPU Counters



## NEW! Simplified Experiments



- Utilization and bottleneck experiments for each unit
- GPU Bottleneck experiment
  - Adds all bottleneck and utilization experiments
  - Expert system analyzes the results
- Exposed via NVPerfAPI

## What is NVPerfHUD?

---

- Direct3D PERFORMANCE Heads Up Display
  - Overlay graphs and debugging tools on top of your application
  - Interactive tools for debugging and performance tuning
- 4 different HUDs
  - Performance Dashboard
  - Debug Console
  - Frame Debugger
  - Frame Profiler (New in 4.0)

## How to use it

---

- Drag and drop your application onto the NVPerfHUD icon
- Run through your application as you normally do until you find:
  - Functional problems: use the Frame Debugger
  - Performance problems: use the Dashboard graphs and Frame Profiler

## How do I use NVPerfKit counters?

---

- PDH: Performance Data Helper for Windows
  - Win32 API for exposing performance data to user applications
  - Standard interface, many providers and clients
  - Sample code and helper classes provided in NVPerfSDK
- Perfmon: (aka Microsoft Management Console)
  - Win32 PDH client application
  - Perfmon's sampling frequency is low (1X/s)
  - Displays PDH based counter values:
    - OS: CPU usage, memory usage, swap file usage, network stats, etc.
    - NVIDIA: all of the counters exported by NVPerfKit
- Good for rapid prototyping

## NEW! NVPerfAPI

---

- NVIDIA API for easy integration of NVPerfKit
  - No more enable counters in NVDevCPL, run app separately
  - No more lag from PDH
- Simplified Experiments
  - Targeted, multipass experiments to determine GPU bottleneck
  - Automated analysis of results to show bottlenecked unit
- Use cases
  - Real time performance monitoring using GPU and driver counters, round robin sampling
  - Simplified Experiments for single frame analysis

## Solutions to common bottlenecks

---

- CPU Bound?
  - In your code:
    - VTune...VTune...VTune... Don't assume!
    - LOD all calculations: Physics, animation, AI, you name it!
  - In driver code:
    - Create all resources up front: textures, VBs, IBs, shaders
    - Reduce locking resources on the fly (discard VBs/IBs, don't write to a surface the GPU is reading from)
    - Create bigger batches: texture atlas, stitch strips together with degenerates
    - Vertex shader constants = lookup table for matrices
    - Instancing
  - Transferring data to GPU
    - Smallest vertex format possible
      - Remove unnecessary data
      - Use smallest data type possible
    - Derive attributes in vertex shader
    - 16 bit indices

## Solutions to common bottlenecks

---

- IDX Bound, Vertex Shader Bound?
  - Reduce vertex attribute count
    - Compute some attributes
    - Combine attributes (2 2D tex coords per attribute)
  - Use geometry LOD
  - Move invariant calculations to the CPU
  - Use indexed primitives, more cache friendly
  - Don't do unnecessary matrix multiplies
  - Use vertex shader branching to bypass expensive calculations
  - Use NVShaderPerf!

## Solutions to common bottlenecks

---

- Pixel Shader Bound?
  - Render depth first (no color writes = 2X speed)
  - Prebake complex math into textures
  - Move per pixel calculations to the vertex shader
  - Use partial precision where possible, try it you may like the result
  - Avoid unnecessary normalizations
  - Use LOD specific pixel shaders
  - Use NVShaderPerf!

## Solutions to common bottlenecks

---

- Texture bound?
  - Prefilter textures to reduce size
  - Mipmap on any texture/surface that might be minified
  - Compressed textures
  - Use float textures only when needed

## Solutions to common bottlenecks

---

- Frame buffer bound?
  - Render depth first (no color writes = 2X speed)
  - Only use alpha blending when necessary
  - Use alpha test
  - Disable depth writes when possible
  - Avoid clearing the color buffer if you touch every pixel (but do clear Z)
  - Render front to back to get better z culling
  - Use float textures only when needed