De JurassicFortran à Fortr'&Furious

Fortran & GPU

Vincent Lafage

IJCLab, Laboratoire de Physique des 2 Infinis Irène Joliot-Curie
CNRS/IN2P3 & Université Paris-Saclay, Orsay, France

vendredi 13 juin 2025

Has Fortran a future ?

https://doi.org/10.1016/0010-4655(85)90086-4

Computer Physics Communications 38 (1985) 199–210
North-Holland, Amsterdam

HAS FORTRAN A FUTURE?

Michael METCALF

Data Handling Division, CERN, CH-1211 Geneva 23, Switzerland

For over 25 years FORTRAN has dominated all other programming languages in the field of scientific and engineering computation. Although much denigrated by computer-science purists, it has consistently shown itself to be attractive to scientific users because its basic simplicitly and power of expression appeal to non-specialists. Can this situation continue? Will the introduction of FORTRAN 77 lead to an upsurge in the use of the language, providing it with momentum sufficient to carry it through to the end of the decade? Shall we witness a conflict between FORTRAN 8x and ADA?

This lecture will take stock of the present status of FORTRAN and describe its likely development, before going on to speculate on possible trends until the turn of the century.

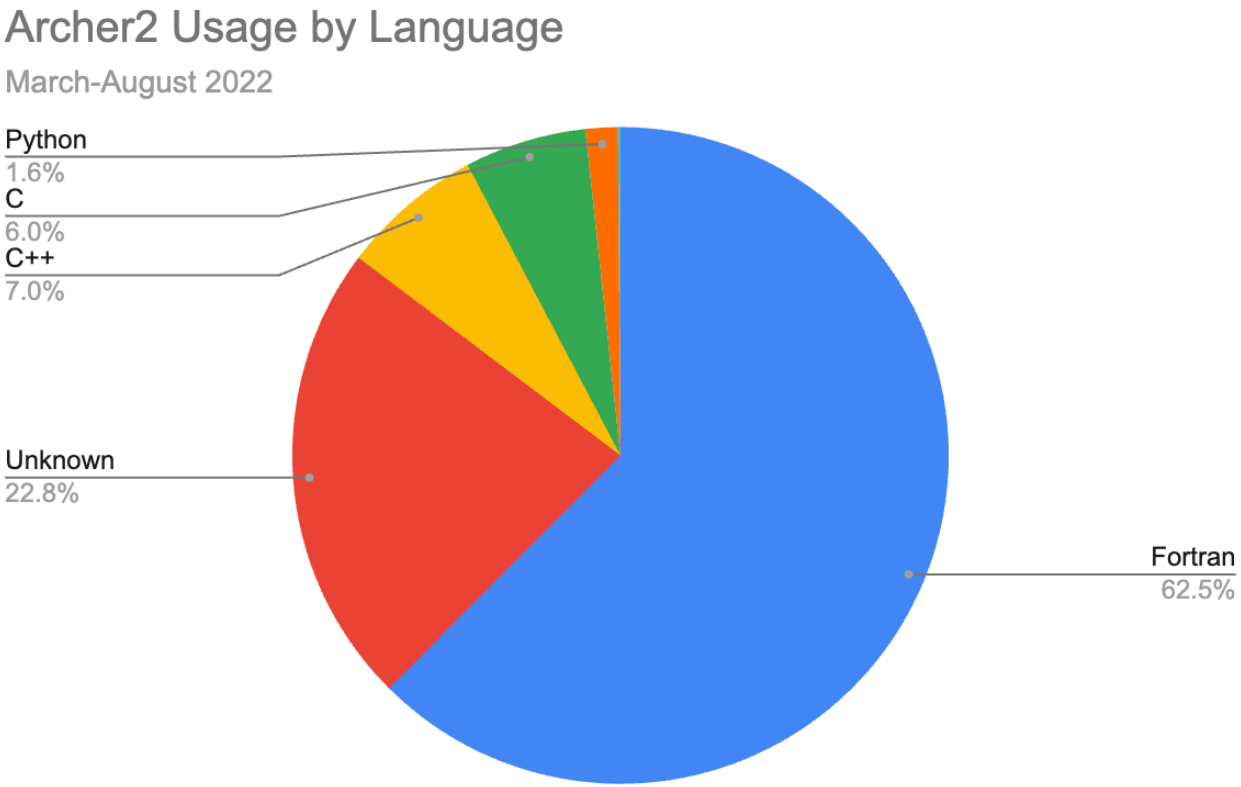
1. FORTRAN today

FORTRAN has a long if not always glorious history. As the first high-level programming language [1] it spread rapidly in those non-commercial application areas for which it was well or reasonably well suited, if only for a lack of competition. Although in the meantime other languages suited for scientific use have appeared, they have failed to attract the same following, possibly because they not only had to compete with an already well established product, but also themselves lacked the twin features of FORTRAN which has kept it in the lead so long – ease of use and between the old 1965 standard and the new standard. Many vendors offer only FORTRAN 77 compilers, or provide, but no longer support, compilers based on the old standard. A survey of European IBM sites I conducted at the beginning of 1984 [4] showed that about two-thirds of the use of FORTRAN was with the VS compiler, though that may hide some use of FORTRAN 66 as it is contained in that compiler as an option. On the CERN mainframes the use of FORTRAN 77 in what is claimed to be an ultra-conservative community has reached 40%, and all new programs will be written to the new standard. On the smaller CERN machines, FORTRAN 77 is used exclu-

Has Fortran a future ?

"There are only two kinds of languages :
the ones people complain about and the ones nobody uses".
(Bjarne Stroustrup)

https://www.stroustrup.com/quotes.html



https://cpufun.substack.com/p/is-fortran-a-dead-language

Historique synoptique

	66	77	90	95	03	08	18/23	28

Procedural		ELSE
		ENDIF
		ENDDO
		DO WHILE
			SELECT CASE			BLOCK		

Modular	COMMON	INCLUDE	MODULE TYPE					
Inheritance					EXTENDS			
Dynamic			ALLOCATE POINTER		*			
Functional	passing subprogram argument		RECURSIVE INTENT()	PURE ELEMENTAL			SIMPLE	
Concurrent			array exp	FORALL		DO CONCURRENT CONTIGUOUS		
Distributed						coarray	TEAM	
Generic	preprocessing	intrinsic	INTERFACE					template

Fortran patrimonial ? SPAG

Portez votre sagesse antique en `Fortran` moderne...

- ...ce n'est pas simplement convertir en *free format*
  au-delà de METCALF's `convert`

  cf `https://twister.caps.ou.edu/ARPS/coding/convert.f90` ou

  `https://hepd.pnpi.spb.ru/docs_html/Fortran90/WWW/f90/convert.f90`
- ...c'est une opportunité de blinder votre code
  - expressivité
  - résilience
  - clarté

Fortran patrimonial ? une recette

- Version Control System : `git`, `hg`…
- Build : `Makefile`, `cmake`, `fpm`…
- tout déclarer (`implicit none`)
- *linters* (`ftnchek`) pour Fortran 77, (`fortitude`, `camfort`) pour Fortran Moderne
- analyseur de complexité cyclomatique (`lizard`)
- test de duplication de code (`simian`)
- *unspagghetiffy* (`SPAG` *cf* `https://polyhedron.com/?product=plusfort`)
- convertir chaque `COMMON` en `include`
- convertir en *free format* (utiliser `convert` de M Metcalf) `https://dl.acm.org/doi/pdf/10.1145/192115.192139` `https://groups.google.com/g/comp.lang.fortran/c/C4Rth_-66is?pli=1`
- convertir chaque `include` en `module`
- convertir les variables statiques initialisées (`save`) À VOCATION CONSTANTE en `parameter`
- déclarer les `intent` de tous les paramètres de procédure
- déclarer `pure` toutes les procédures possibles
- déclarer `elemental` toutes les procédures possibles
- généricer le `kind` des variables flottantes
- utiliser la forme générique de toutes les fonctions
- convertir les vieilles boucles `do` … `end do` en
  - appel de fonctions `elemental` / boucle `forall` / boucle `do concurrent`
- utiliser des fonctions plus appropriées :
  - `dot_product`, `norm2`, `matmul`, `reshape`, `transpose`
- Documentation : Doxygen, Ford `https://github.com/Fortran-FOSS-Programmers/ford`

Fortran 2003 C (thin) bindings

```
!    Interfaces C (private)
interface
   pure function to_dec32(x) bind(C, name="to_dec32")
      import
      real(c_float), value :: x
      real(c_float)        :: to_dec32
   end function to_dec32

   pure function dec32add(x,y) bind(C, name="dec32add")
      import
      real(c_float), value :: x, y
      real(c_float)        :: dec32add
   end function dec32add

   pure function dec32sub(x,y) bind(C, name="dec32sub")
      import
      real(c_float), value :: x, y
      real(c_float)        :: dec32sub
   end function dec32sub

   pure function dec32mul(x,y) bind(C, name="dec32mul")
      import
      real(c_float), value :: x, y
      real(c_float)        :: dec32mul
   end function dec32mul

   pure function dec32div(x,y) bind(C, name="dec32div")
      import
      real(c_float), value :: x, y
```

Fortran 2003 C thick bindings

```
module decimal_interface
   use, intrinsic :: iso_fortran_env, only: real128
   use, intrinsic :: iso_c_binding, only: c_float, c_double, c_long_double, c_ptr, c_loc
   implicit none

   private
   public :: c_float, c_double, real128
   public :: dec32, dec64, dec128
   public :: dec32_show, dec64_show, dec128_show
   public :: operator(+), operator(-), operator(*), operator(/), operator(**)

   type, bind(C) :: dec32
      private
      real(c_float) :: data
   end type dec32

   interface dec32
      module procedure dec32_of_real, dec32_of_string
   end interface dec32

   interface operator(+)
      module procedure dec32_add
   end interface

   interface operator(-)
      module procedure dec32_sub
   end interface
```

Missing In `Fortran`

- héritage multiple ⇒ Facade Pattern
- first-class functions ⇒ `real, external, pointer :: f_ptr.`
  Alternative : `procedure (f), pointer :: f_ptr`
  `https://gcc.gnu.org/onlinedocs/gfortran/Working-with-C-Pointers.html`
- gestion d'exception (au-delà des exceptions IEEE 754)
- `assert`
- généricité (polymorphisme paramétrique) ⇒ `Fortran202Y`
  `https://github.com/j3-fortran/generics/tree/main`
- `const` ? (≠ `parameter` ∼ `consteval`) ⇒ …, `intent (in) ::` ou …, `protected ::`
- `C` binding to variadic functions
- coroutines
- …

- MOORE's Law : Gordon MOORE's observation (1965)
  « Cramming More Components onto Integrated Circuits. » :
  The number of transistors
  in a dense integrated circuit (IC)
  doubles about every two years.
  (even before microprocessors)

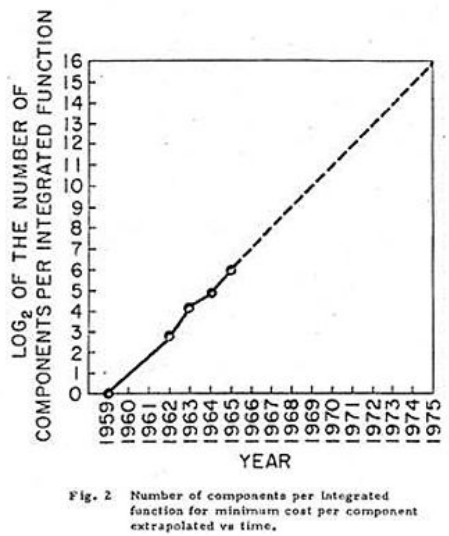


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

+ registers
+ memory cache
+ processor instructions
+ bus size (4 bits → 64 bits)
+ memory management (MMU)
+ processing units (one, then many ALU/FPU, vector ALU/FPU…)
+ pipeline depth (superscalars *cf* Pentium ca 1993)
+ complex branch predictor / out-of-order execution unit

- **Heat/Power Wall** : $\mathcal{P} = \alpha \cdot C \cdot {V_{dd}}^2 \cdot f + V_{dd} \cdot I_{st} + V_{dd} \cdot I_{leak}$
- **Frequency Wall** : « Free lunch is over » (already for 15 years, almost 20 years)
- 1971 ⇒ 10 µm, 2012 ⇒ 22 nm, 2014 ⇒ 14 nm, 10 nm in (slow) progress (Intel). TSMC, Samsung : 7 nm, 5 nm factories. 3 nm and beyond down to 1.4 nm in Intel roadmap. Tunnel effect ⇒ **Quantum Wall**
- **Money Wall**

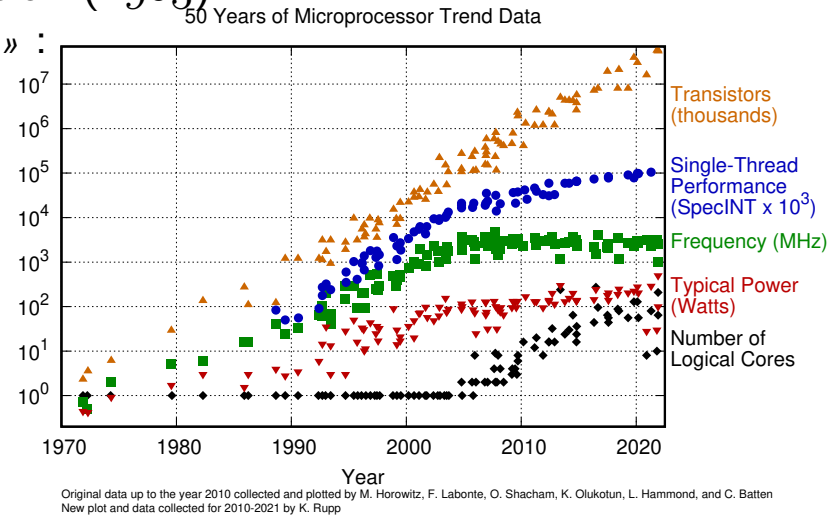
Why parallelize ? end of MOORE's law

- MOORE's Law : Gordon MOORE's observation (1965)

  « Cramming More Components onto Integrated Circuits. » :

  The number of transistors
  in a dense integrated circuit (IC)
  doubles about every two years.

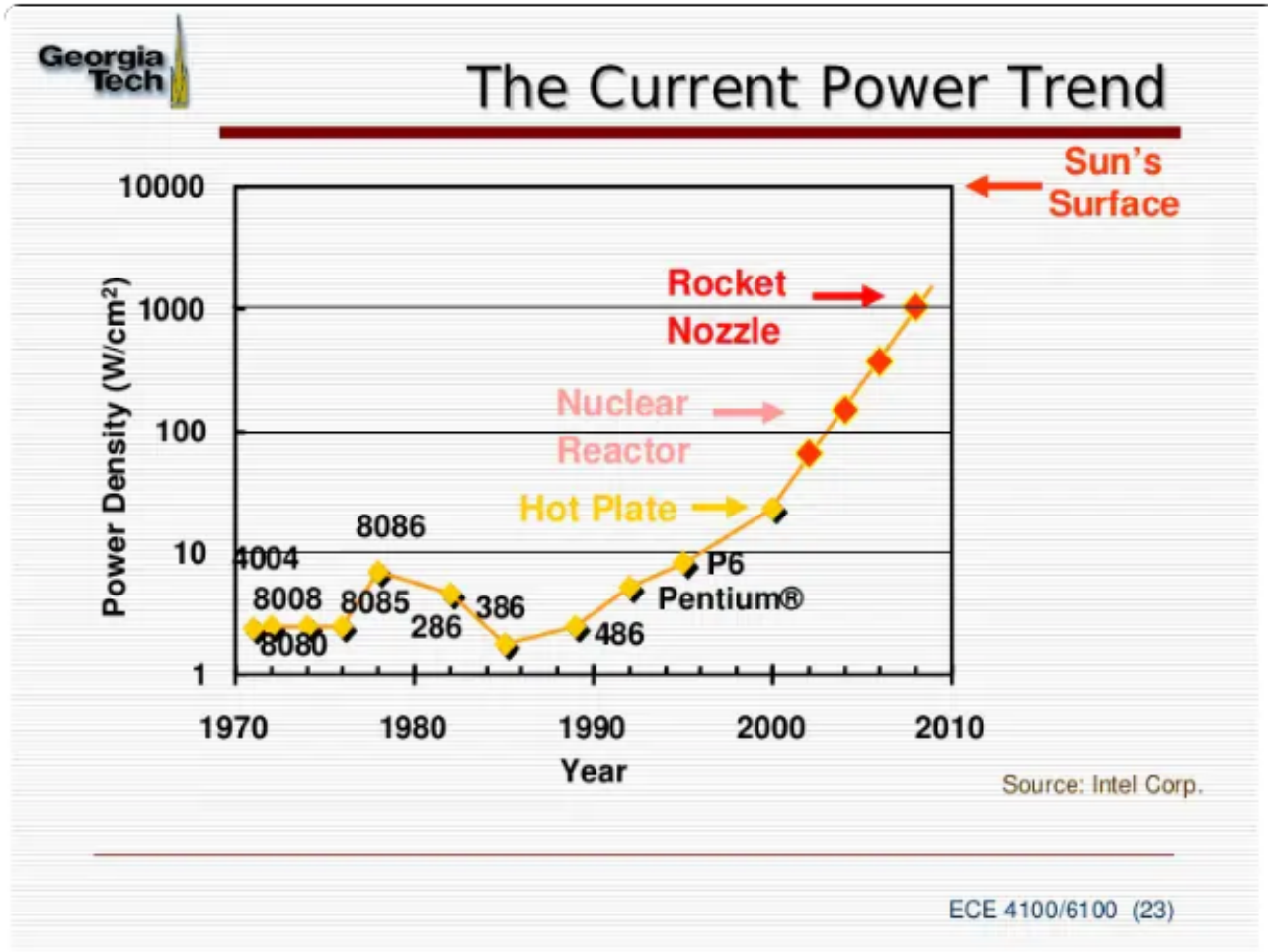
  (even before microprocessors)



  + registers
  + memory cache
  + processor instructions
  + bus size (4 bits → 64 bits)
  + memory management (MMU)
  + processing units (one, then many ALU/FPU, vector ALU/FPU…)
  + pipeline depth (superscalars *cf* Pentium ca 1993)
  + complex branch predictor / out-of-order execution unit

- **Heat/Power Wall** : $\mathcal{P} = \alpha \cdot C \cdot {V_{dd}}^2 \cdot f + V_{dd} \cdot I_{st} + V_{dd} \cdot I_{leak}$
- **Frequency Wall** : « Free lunch is over » (already for 15 years, almost 20 years)
- 1971 ⇒ 10 µm, 2012 ⇒ 22 nm, 2014 ⇒ 14 nm, 10 nm in (slow) progress (Intel). TSMC, Samsung : 7 nm, 5 nm factories. 3 nm and beyond down to 1.4 nm in Intel roadmap. Tunnel effect ⇒ **Quantum Wall**
- **Money Wall**

Why parallelize ? Frequency/Power Wall



Introduction to Multicore architecture, Tao ZHANG – Oct. 21, 2010

Why parallelize ? in the era of climate change

Information technologies : growing part of a rare, expensive & dirty energy.
1.6 MW for the first room of IN2P3 Computing Centre : 0,5 to 1 M€/yr

Moving from PFlops to Exascale requires a breakthrough...

- moving to a **better W/MIPS ratio** (or W/MFLOPS) :
  Intel XScale [1], 600 MHz, 0.5 W
  5 × slower, 80 × cheaper in energy !
- **reduce frequency**, using more cores

« The number of computations per joule of energy dissipated doubled about every 1.57 years. »

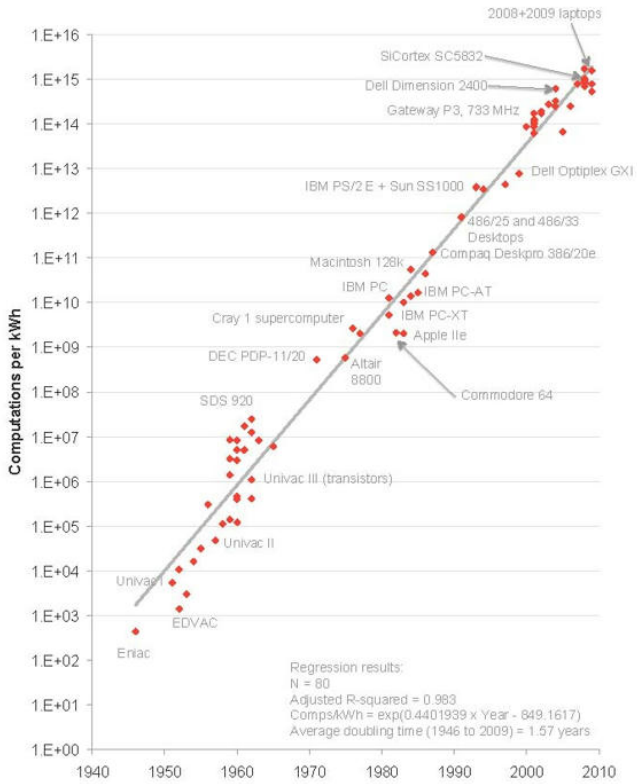
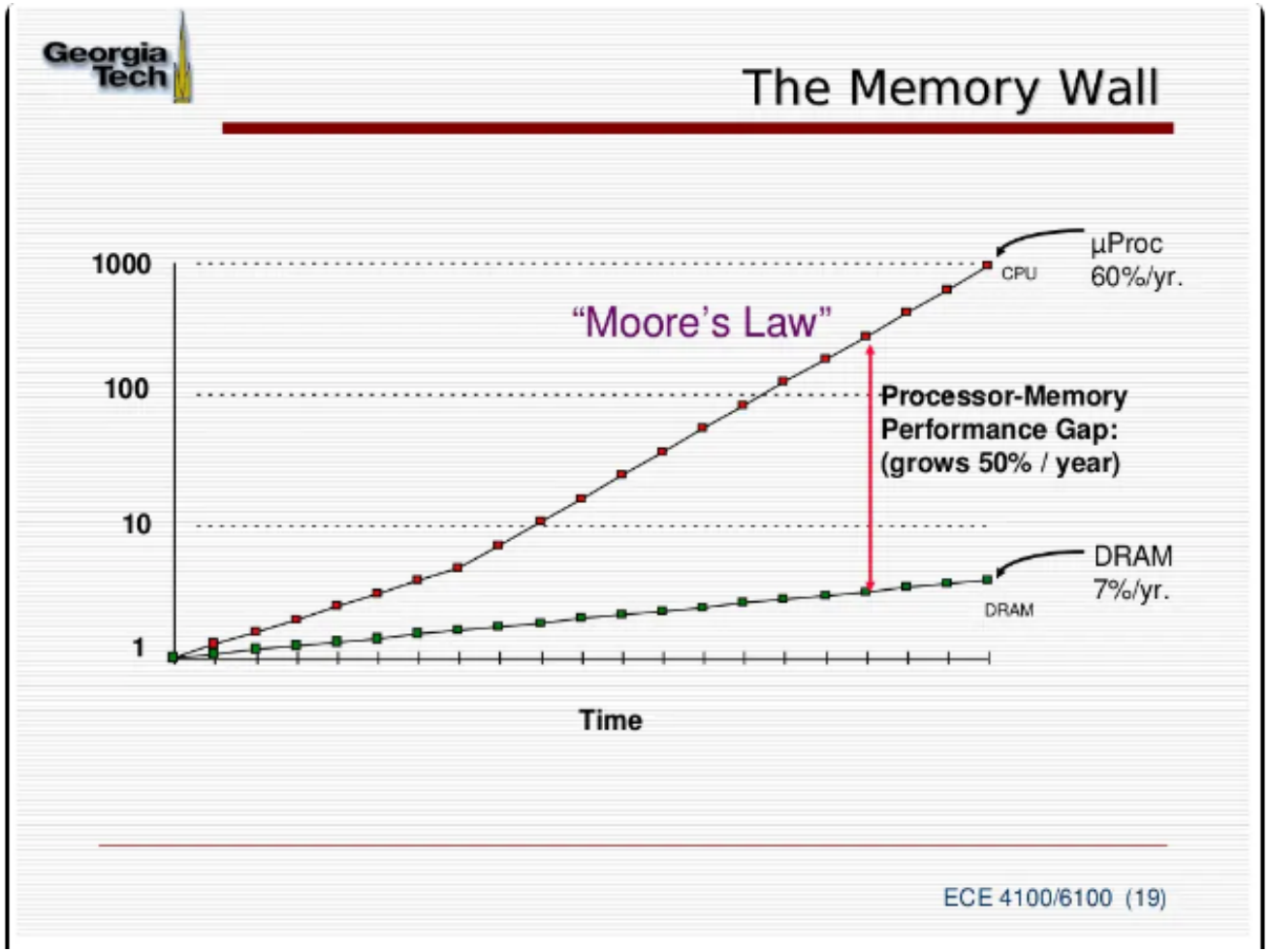
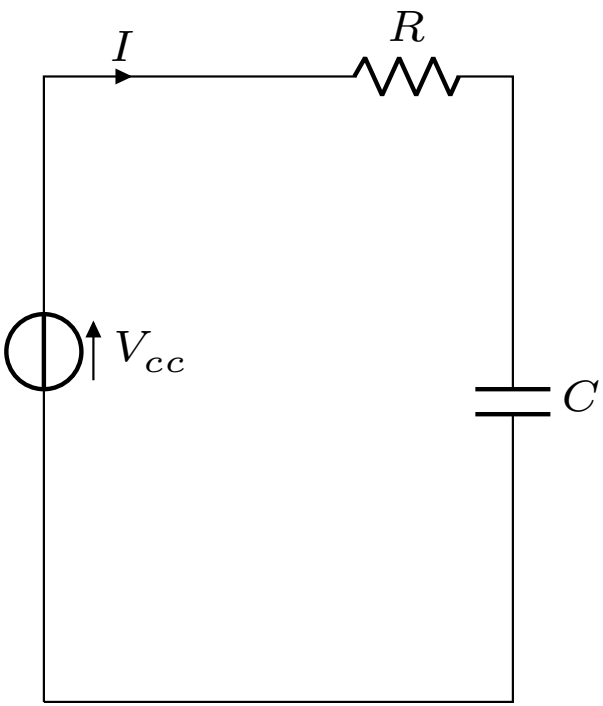


Figure – Koomey's law, 2010

Why parallelize ? Yet another Wall...



Introduction to Multicore architecture, Tao ZHANG – Oct. 21, 2010



Data is moved through wires

Wires/memory behave like an RC circuit

Trade-off :

- Longer response time $\tau = RC$ ("latency")
- Higher current I ($\Rightarrow$ more power)

Physics says :

Communication is slow, power-hungry, or both

Hierarchy of memories

- Small amount of fast memory close to CPU
- Large amount of slow memory far from CPU

CPU register « Level 1 cache « Level 2 cache « Level 3 cache « Main memory « Disk « Internet

Why parallelize ? Memory Wall

We must feed the CPU $\Rightarrow$ some problems will be **memory bound**.
The distinction between **memory bound** and **CPU bound** algorithms can often be related to their **arithmetic intensity** :
for N-sized problem, how many operations ?

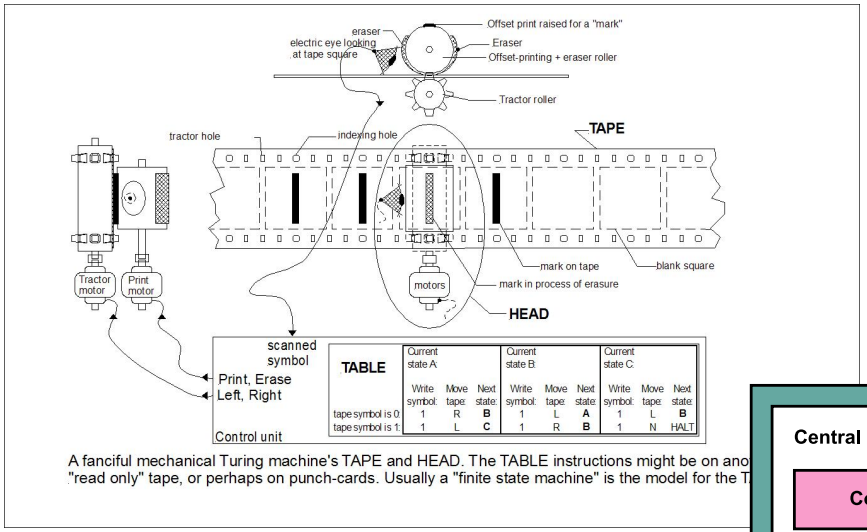
- dotproducts : $\mathcal{O}(N)$ data, $\mathcal{O}(N)$ ops
  convolution
- matrix-vector products : $\mathcal{O}(N(N+1))$ data, $\mathcal{O}(N^2)$ ops
- matrix-matrix products : $\mathcal{O}(2N^2)$ data, $\mathcal{O}(N^3)$ ops
  matrix decomposition/factorisation/reduction for system inversion, diagonalisation, Fourier/Bessel transform……

"If the only tool you have is a hammer, you tend to see every problem as a nail."

Maslow*'s gavel*

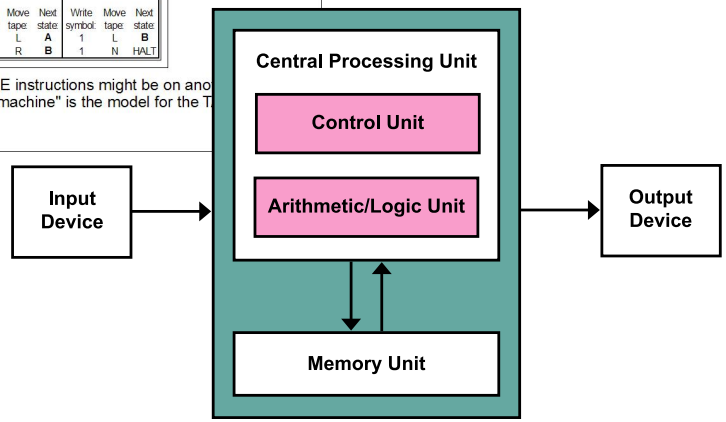
$\Rightarrow$
"If the only tool you have is a GPU, you tend to see every problem as a matrix product."

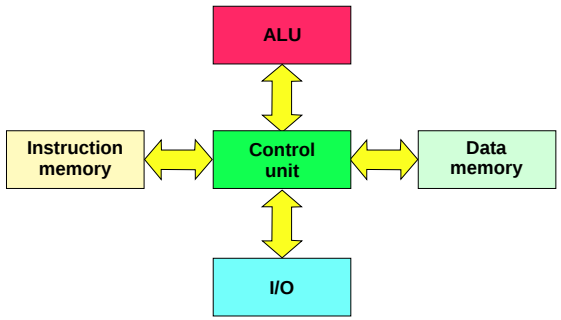
Architecture



- Turing Machine
- von Neumann architecture (Princeton architecture)

  $\Rightarrow$ von Neumann bottleneck
- Harvard architecture





Échelles du parallélisme

- Processus (dans des environnements bien séparés) :
  - problèmes dits *« embarassingly parallel »*
    — traitement distribué par événement (physique des particules) sur une grille
  - communication inter-processus *à la Unix*
    — pipe
    — mémoire partagée
    — sémaphore
  - avec des ordinateurs séparés et passage de messages à travers le réseau (systèmes à mémoire distribuée) :
    — `PVM` *« Parallel Virtual Machine »* 1989
    — `MPI` *« Message Passing Interface »* 1994

- Threads (*alias « processus légers »*)
  - overhead réduits : *par rapport à un processus* — `C/C++ pthread` *« POSIX thread »* 1995
    — tâche `Ada` / objets protégés
  - Shared Memory sur SMP *« Symmetric Multi Processing »* machines :
    — `OpenMP` *« Open Multi Processing »* 1997.

- Vectorisation :
  - même les threads ont trop de surcharge lorsque les tâches concurrentes sont élémentaires :
    ⇒ nous ne pouvons utiliser aucun thread CPU pour la somme individuelle ou les produits de nombreuses données ⇒ we can use no CPU thread for individual sum or products of many data
  - ILP *« Instruction Level Parallelism »*
    — processeur vectoriel *« à la Cray »* : SIMD *« Single Instruction, Multiple Data »*

    — processeur moderne avec SIMD à longueur fixe

      `MMX` *« MultiMedia eXtension »* 1997 Pentium P5
      `SSE` *« Streaming Extension »* 1999 Pentium ///
      `AVX` *« Advanced Vector eXtension »* 2008-2011
      `AVX512` 2013-2017

Comment vectoriser ? ranger les données

Nous connaissons *le parallélisme au niveau des bits* :
nous pouvons AND ou OR 8, 16, 32, 64 bits en une seule instruction
De la même manière, pour que notre instruction vectorielle fonctionne de manière fluide, nous devons aligner notre vecteur de données sur les limites de la mémoire.

Nous avons intérêt à ordonner les données de même nature comme un tableau (aligné) (contigu)
$\Rightarrow$ pour la contiguïté des données nous préférons une

structure de tableaux

à la

collection (tableau) d'objets (structure)

orientée objet classique.

Comment vectoriser ?

La vectorisation, un processus d'industrialisation :
considèrez votre calcul comme une ligne de production

Prise en tenaillle de la performance :

- paralléliser d'en haut,
- vectoriser d'en bas

... + garder le cache chaud !

Comment vectoriser ?

1. créer manuellement le code vectoriel avec des instructions de bas niveau dédiées (intrinsèques)
2. utiliser des bibliothèques vectorielles
3. laisser la main au compilateur, à supposer qu'on puisse
   1. exprimer le parallélisme des données (aspect de haut niveau)
      $\Rightarrow$ mettre à profit la puissance des tableaux (*à la Matlab / Fortran 90*)
   2. donner des indices aux compilateur quant à la contiguité des données & leur alignement (aspect de bas niveau)

Et puis, comment vérifier qu'elle a bien été vectorisée ?

- flagS du compilateur pour signaler la vectorisation
- vérifier l'assemblage résultant (yuk !)
- `maqao`
- ...indirectement, `perf`

Comment vectoriser ?

Les nombres complexes : une vectorisation symbolique
... qui fait obstacle à la vectorisation pratique !

$$(a+ib)\times(c+id) = (ac-bd)+i(ad+bc)$$

On veut un complexe de vecteurs plutôt qu'un vecteur de complexes.

$$\begin{array}{rcl}
(a_1+ib_1)\times(c_1+id_1) & = & (a_1c_1-b_1d_1)+i(a_1d_1+b_1c_1)\\
(a_2+ib_2)\times(c_2+id_2) & = & (a_2c_2-b_2d_2)+i(a_2d_2+b_2c_2)\\
(a_3+ib_3)\times(c_3+id_3) & = & (a_3c_3-b_3d_3)+i(a_3d_3+b_3c_3)\\
(a_4+ib_4)\times(c_4+id_4) & = & (a_4c_4-b_4d_4)+i(a_4d_4+b_4c_4)
\end{array}$$

Code vectorisable ...

...

```
        resultat_my[i] = my_cbrtf (harvestcb[i]);
    }
    stop = __rdtsc();
    delay  = stop-start;
    printf ("MCbrt:           %10lu\t%9.3f tick/iter\n", delay, ((float) delay)/((float) size));
    prepare (harvestcb, resultat_my, size, "MCbrt", my_cbrtf);

    start = __rdtsc();
    my_vec_cbrtf (harvestcb, resultat_myv, size);
    stop = __rdtsc();
    delay  = stop-start;

    printf ("VCbrt:           %10lu\t%9.3f tick/iter\n", delay, ((float) delay)/((float) size));
//    prepare (harvestcb, resultat_myv, size, "VCbrt", my_cbrtf);

    start = __rdtsc();
#pragma omp for simd aligned (harvestcb, resultat_pow) safelen(64)
```

...

Vectorisation

Yapluka ?

```
gfortran -Ofast -march=native -mtune=native -mavx512f
    -fopenmp -fopenmp-simd -ftree-vectorize -fopt-info -ftree-vectorizer-verbose=1
    -mrecip --param=ssp-buffer-size=4
    -g -Wall -Wextra -Wshadow -Wconversion -Wpedantic
    ../cbrt_mod.f90 ../cbrt_validate.f90 -o validate_cbrt -lm && taskset --cpu-list 0 ./valida

nvfortran -Ofast -march=native -mtune=native -mavx512f
    -fopenmp -g -Wall -Wextra
    ../cbrt_mod.f90 ../cbrt_validate.f90 -o validate_cbrt -lm && taskset --cpu-list 0 ./valida
```

Vectorisation

Est-ce que ça marche ?

```
objdump --disassemble=nw_cbrt validate_cbrt
```

```
0000000000401c30 <my_cbrtf>:
  401c30:       c5 f8 28 f0             vmovaps %xmm0,%xmm6
  401c34:       c5 fa 10 25 24 d5 00    vmovss 0xd524(%rip),%xmm4        # 40f160 <_IO_stdin_used+0x1e0>
  401c3b:       00
  401c3c:       c5 fa 10 15 1c d6 00    vmovss 0xd61c(%rip),%xmm2        # 40f260 <_IO_stdin_used+0x2e0>
  401c43:       00
  401c44:       c5 c8 54 fc             vandps %xmm4,%xmm6,%xmm7
  401c48:       c5 f9 7e f9             vmovd  %xmm7,%ecx
  401c4c:       c5 f9 7e f8             vmovd  %xmm7,%eax
  401c50:       c1 f9 17                sar    $0x17,%ecx

...

  401c92:       c4 e2 79 a9 15 c9 d5    vfmadd213ss 0xd5c9(%rip),%xmm0,%xmm2        # 40f264 <_IO_stdin_used+0x2e4>
  401c99:       00 00
  401c9b:       c4 e2 79 a9 15 c4 d5    vfmadd213ss 0xd5c4(%rip),%xmm0,%xmm2        # 40f268 <_IO_stdin_used+0x2e8>
  401ca2:       00 00
  401ca4:       c4 e2 79 a9 15 bf d5    vfmadd213ss 0xd5bf(%rip),%xmm0,%xmm2        # 40f26c <_IO_stdin_used+0x2ec>
  401cab:       00 00
  401cad:       c5 ea 59 ea             vmulss %xmm2,%xmm2,%xmm5
  401cb1:       c5 f8 28 ca             vmovaps %xmm2,%xmm1
  401cb5:       c4 e2 79 9d cd          vfnmadd132ss %xmm5,%xmm0,%xmm1
  401cba:       c5 f2 59 da             vmulss %xmm2,%xmm1,%xmm3
  401cbe:       c5 ea 58 ca             vaddss %xmm2,%xmm2,%xmm1
  401cc2:       c4 e2 79 99 e9          vfmadd132ss %xmm1,%xmm0,%xmm5
  401cc7:       c5 d2 53 cd             vrcpss %xmm5,%xmm5,%xmm1
  401ccb:       c5 f2 59 ed             vmulss %xmm5,%xmm1,%xmm5
  401ccf:       c5 f2 59 ed             vmulss %xmm5,%xmm1,%xmm5
  401cd3:       c5 f2 58 c9             vaddss %xmm1,%xmm1,%xmm1
  401cd7:       c5 f2 5c cd             vsubss %xmm5,%xmm1,%xmm1
  401cdb:       c5 e2 59 c9             vmulss %xmm1,%xmm3,%xmm1
  401cdf:       c5 f2 58 d2             vaddss %xmm2,%xmm1,%xmm2
  401ce3:       c5 ea 59 da             vmulss %xmm2,%xmm2,%xmm3
```

http://maqao.org/

MAQAO (Modular Assembly Quality Analyzer and Optimizer) is a performance analysis and optimization framework operating at binary level with a focus on core performance.

MAQAO mixes both dynamic and static analyses based on its ability to reconstruct high level structures such as functions and loops from an application binary. [2]

```
maqao.intel64 cqa fct-loops=my_cbrtf conf=hint,expert --output-format=html --follow-calls=inline s

Section 2.1.1: Binary loop #13
==============================

The loop is defined in:
 - /usr/local/lafage0/IJCLab/Reprises/my_float_cuberoot.c:52-70,93-121
 - /usr/local/lafage0/IJCLab/Reprises/speedcbrtf.c:64


The related source loop is multi-versionned.
43% of peak computational performance is used (13.95 out of 32.00 FLOP per cycle (GFLOPS @ 1GHz))

Vectorization
-------------
Your loop is fully vectorized, using full register length.

All SSE/AVX instructions are used in vector version (process two or more data elements in vector registers).


Execution units bottlenecks
---------------------------
Performance is limited by:
 - execution of FP add operations (the FP add unit is a bottleneck)
 - execution of FP multiply or FMA (fused multiply-add) operations (the FP multiply/FMA unit is a bottleneck)

By removing all these bottlenecks, you can lower the cost of an iteration from 19.50 to 17.00 cycles (1.15x speedup).

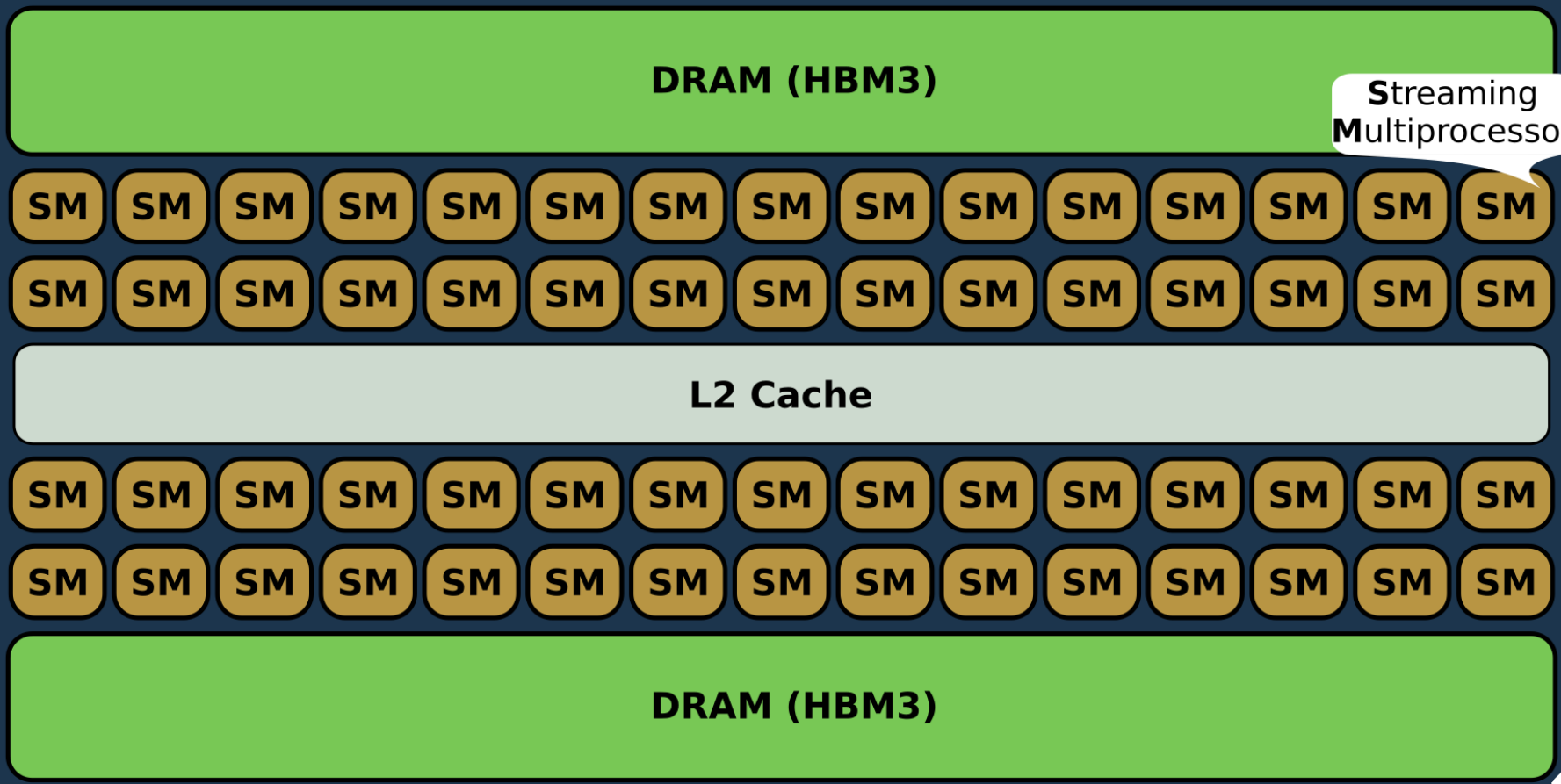
Workaround(s):
 - Reduce the number of FP add instructions
 - Reduce the number of FP multiply/FMA instructions


FMA
---
```

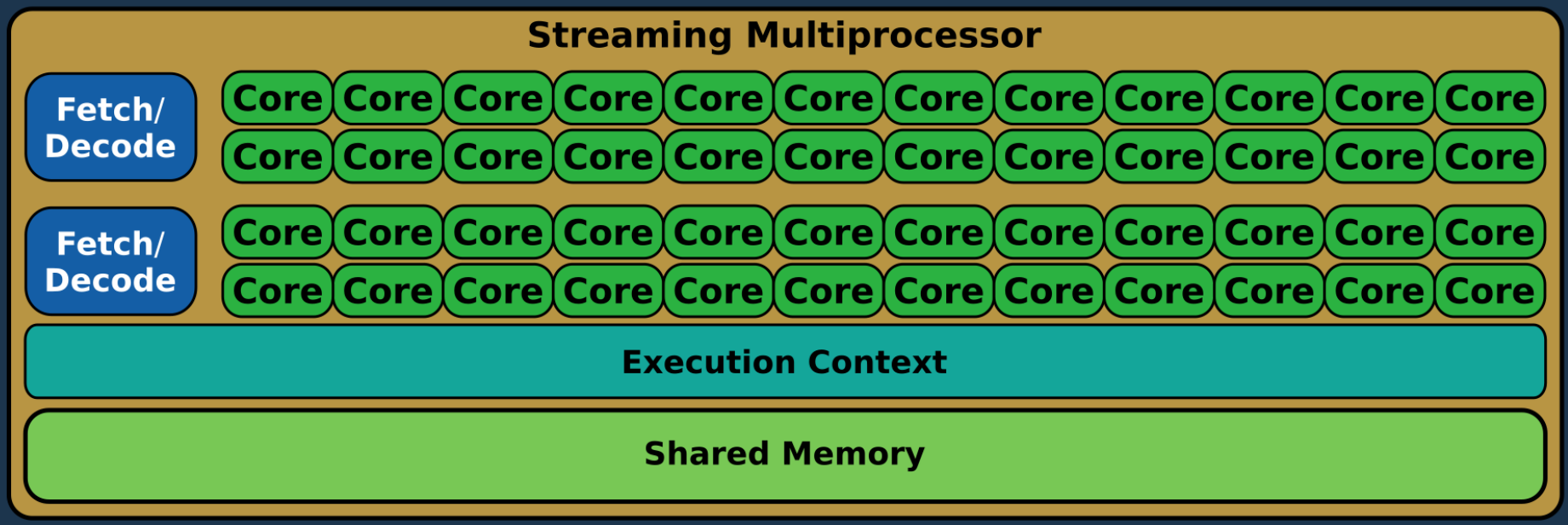
```
...
VFMADD213PS %YMM1,%YMM4,%YMM7
VBROADCASTSS 0x2a9ea(%RIP),%YMM1
VFMADD213PS %YMM1,%YMM4,%YMM7
VBROADCASTSS 0x2a9e0(%RIP),%YMM1
VFMADD213PS %YMM1,%YMM4,%YMM7
VMULPS %YMM7,%YMM7,%YMM1
VMULPS %YMM7,%YMM1,%YMM1
VBROADCASTSS 0x2a9ce(%RIP),%YMM5
VMOVAPS %YMM5,%YMM12
VFMADD213PS %YMM4,%YMM1,%YMM12
VRCPPS %YMM12,%YMM6
VPADDD %YMM15,%YMM9,%YMM13
VSUBPS %YMM1,%YMM4,%YMM1
VMOVUPS 0x42d3a0,%YMM11
VFMSUB213PS %YMM11,%YMM6,%YMM12
VMULPS %YMM6,%YMM1,%YMM1^^I1^^I0.50
VBROADCASTSS 0x2a98c(%RIP),%YMM6
VBROADCASTSS 0x2a987(%RIP),%YMM3
VFMADD213PS %YMM3,%YMM13,%YMM6
VBROADCASTSS 0x2a97d(%RIP),%YMM9
VFMADD213PS %YMM9,%YMM13,%YMM6
VBROADCASTSS 0x2a973(%RIP),%YMM9
VFMADD213PS %YMM9,%YMM13,%YMM6
VFMSUB213PS %YMM7,%YMM7,%YMM12
VFNMADD213PS %YMM7,%YMM1,%YMM12
VMULPS %YMM6,%YMM6,%YMM1
VMULPS %YMM6,%YMM1,%YMM1
...
```

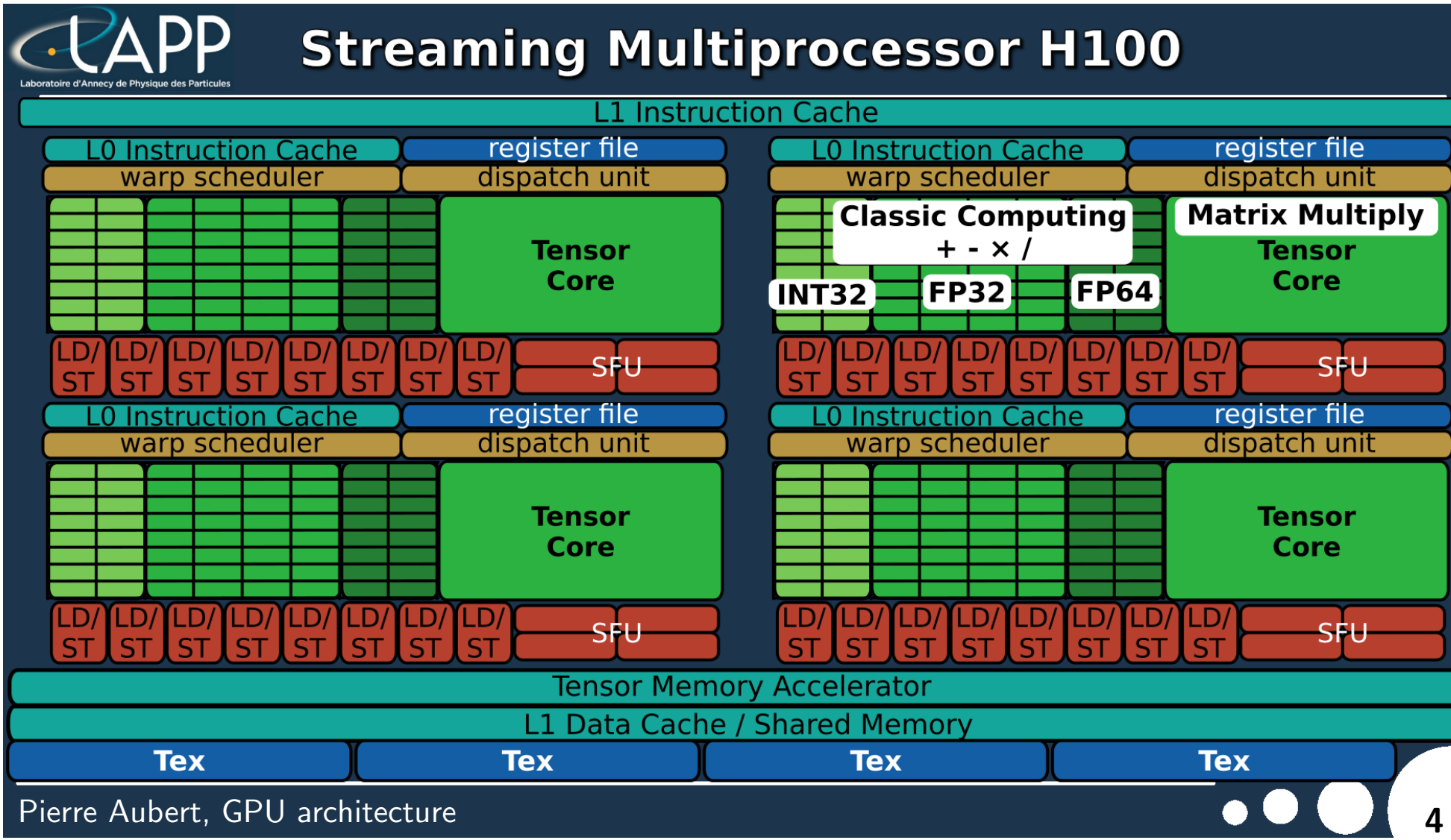
- V Vector instruction...
- SS ...Scalar Single precision
- PS ...Packed Single precision

GPU Architecture



Streaming Multiprocessor




LAPP
Laboratoire d'Annecy de Physique des Particules
Streaming Multiprocessor H100
L1 Instruction Cache
L0 Instruction Cache
warp scheduler
register file
dispatch unit
Tensor Core
Classic Computing
+ - × /
INT32
FP32
FP64
Matrix Multiply
Tensor Core
LD/ ST
SFU
Tensor Memory Accelerator
L1 Data Cache / Shared Memory
Tex
Pierre Aubert, GPU architecture
4

Compute terminology translation guide

CPU (with an x86 bias)	NVidia CUDA	Khronos (OpenCL, SYCL, Vulkan…)
SIMD lane	CUDA core	Processing Element
SIMD vector	Warp	Subgroup
Simultaneous Multi-Threading / Hyper-Threading	Hardware multithreading	*Not named, hidden inside PE/CU model*
Advanced Matrix eXtensions, Neural Processing Unit	Tensor core	*No standard name (only usable via Vulkan vendor extensions)*
No equivalent, ray tracing is not special	RTX core	*Not named, exposed via VK_KHR extensions*
Intel Data Streaming Accelerator	Tensor Memory Accelerator (kind of [1])	*Not named, exposed via OpenCL async copy API*
Core	Streaming Multiprocessor	Compute Unit
Core complex / Sub-NUMA cluster	Thread block cluster (kind of [2])	*Not exposed yet*
NUMA node	Device	Device
SIMD registers, with manual [3] spill to caches/RAM	Local memory, with automatic spill to global memory	Private memory, with automatic spill to global memory
L1/L2 cache, with automatic spill to lower caches/RAM	Shared memory	Local memory
No equivalent, all memory is writable	Constant memory	Constant memory
No equivalent, images are not special	Texture memory	*Not named, exposed via image API*
Cache hierarchy + RAM	Global memory	Global memory
SIMD lane	Threads within the same warp	Work-items within the same subgroup
SIMD vector	Warp	Subgroup
Thread	Threads within the same thread block	Work-items within the same work-group
No equivalent, all threads can synchronize with each other	Thread block	Work-group
No equivalent, all threads can synchronize with each other	Thread block cluster	*Not exposed yet*
No equivalent, threads are independent from each other	Grid	NDrange
No equivalent, can spawn threads anytime	Stream	Command queue
No equivalent, can spawn threads anytime	Graph	Command buffer *(only exposed in Vulkan)*
Buffer	Buffer	Buffer
No equivalent, images are not special	Texture	Image
No equivalent, images are not special	Texture	Sampler
RAM	Memory	Device memory
No equivalent, all useful memory is reachable	Unified memory	Shared memory
No equivalent, all useful memory is reachable	Host/pinned memory	Host memory

[1] Both TMA and DSA allow offloading memory copy work from the compute cores, but Intel DSA is more focused on device DMA and Nvidia TMA on VRAM-cache copies.
[2] On a hardware level, it's the same idea, but CUDA only allows some forms of synchronization when threads belong to the same thread block cluster.
[3] Speaking from the perspective of machine code here. Of course, the compiler of most programming langages will automate it for you.

Capitaliser sur les tableaux la chasse aux boucles est ouverte

- opérations sur tableaux : $+$, $-$, $*$ (HADAMARD), $/$ (*element-wise*)
- applications de fonctions sur tableaux (mapping)
  - les fonctions intrinsèques
  - les fonctions `ELEMENTAL` : `PURE` + automap
- réduction de tableaux : `SUM`, `PRODUCT(ARRAY, DIM, MASK)`
- `DOT_PRODUCT(VECTOR1, VECTOR2)`
- `MATMUL`
- `RESHAPE`
- `WHERE (mask) … [ELSEWHERE (mask) …] [ELSEWHERE …] END WHERE`
- …
- `FORALL (variable = from:to{:step})`
- `DO CONCURRENT (variable = from:to{:step})`

Capitaliser sur les tableaux la chasse aux boucles est ouverte

- **Array Expressions**
  - A concise way to perform element-wise operations on arrays.
  - Enables implicit parallelism and enhances readability.
  - Generalizes scalar operations to entire arrays.
  - Takes iterations out of the picture.
- **Forall Loop**
  - A construct for array assignments with explicit iteration.
  - Offers more control than array expressions, with potential for parallel execution.
  - Bridges scalar loops and array operations.
- **Do Concurrent Loop**
  - A modern loop construct designed for parallelism.
  - Allows independent iterations, optimized by compilers for multi-threading or GPU execution.
  - Most powerful, with flexibility for complex computations.
  - The statements within the DO CONCURRENT are executed in order, in each "iteration", but there is no dependence on other iterations. FORALL could have array assignments only, but each assignment needed to be completed by all iterations before the next one executed, effectively creating a "wait for all" after each assignment. Yes, the idea of FORALL was to help with parallelization, but like a lot of High-Performance Fortran, the obsolete variant from which FORALL comes, it was not well thought through and parallelization was much more difficult than it seemed it would.

Capitaliser sur les tableaux la chasse aux boucles est ouverte

- **Array Expressions**
  - `real, dimension(10) :: a, b, c, d`
  - `c = a * b + 2.0` (Combined operations)
  - `d = sqrt(c)` (Elemental ⇒ pure functions)
  - `real, dimension(5,5) :: m`
    `m = m + transpose(m)` (Matrix symmetrization)

- **Forall Loop**
  - `forall (i=1:10) a(i,i) = 0.0` (Set diagonal elements to zero)
  - `integer, dimension(100) :: idx forall (i=1:100, idx(i) > 0) idx(i) = idx(i) - 1`
    (Conditional decrement)

- **Do Concurrent Loop**
  - ```
    real, dimension(100) :: a, x
    do concurrent (i=1:100)
      a(i) = sin(x(i)) * cos(x(i))
      if (a(i) < 0) a(i) = 0.0
    end do
    ```
    (Pure function with logic)
  - ```
    real, dimension(1000) :: c
    do concurrent (i=1:1000) reduction(+:total)
      total = total + exp(c(i))
    end do
    ```
    (Parallel reduction)
  - ```
    real, dimension(10, 10) :: b
    do concurrent (i=1:10, j=1:10) local(tmp)
      tmp = 0.0
      if (i /= j) tmp = 1.0 / real(i-j)
      b(i,j) = tmp
    end do
    ```
    (Local variable and logic)
    ```

- OpenCL
  + passe-partout (y compris hors NVIDIA)
  – Lourd (+ Binding C)
- (Vulkan)
  + passe-partout (y compris hors NVIDIA)
  – Lourd (+ Binding C)
- CUDAFortran
  + bibliothèques cuBLAS, cuFFT, cuRAND
  – passe seulement sur NVIDIA
- OpenACC (directives !$acc)
  – gestion explicite de la mémoire
- OpenMP (directives !$omp)
  – modèle d'*offloading* distinct
- nvfortran
  – passe seulement sur NVIDIA
  + do concurrrent ⇒ rien à faire (vraiment ???)

```
$ nvidia-smi
Fri Jun 13 12:35:50 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 560.35.05              Driver Version: 560.35.05      CUDA Version: 12.6     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA RTX A4000 Laptop GPU    On  |   00000000:01:00.0 Off |                  N/A |
| N/A   61C    P8             17W /   90W |      18MiB /   8192MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A      3899      G   /usr/lib/xorg/Xorg                             4MiB |
+-----------------------------------------------------------------------------------------+
```

- bon module noyau
- ... avec le bon noyau
- ... compatible avec la *compute capability* du GPU

```
module vulkan_types
  use, intrinsic :: iso_c_binding, only: c_int, c_char, c_null_char, c_ptr !, c_loc
  implicit none

  ! Constantes Vulkan (simplifiées)
  integer(c_int), parameter :: VK_SUCCESS = 0
  integer(c_int), parameter :: VK_STRUCTURE_TYPE_APPLICATION_INFO = 0
  integer(c_int), parameter :: VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO = 1
  integer(c_int), parameter :: VK_API_VERSION_1_0 = int(z'00400000', c_int)
  character(kind=c_char), parameter :: &
       defaultAppName = "Vulkan Image Display"//c_null_char

  ! Types Vulkan simulés (seulement les champs nécessaires)
  type, bind(c) :: VkApplicationInfo
    integer(c_int) :: sType
    type(c_ptr) :: pApplicationName
    integer(c_int) :: applicationVersion
    type(c_ptr) :: pEngineName
    integer(c_int) :: engineVersion
    integer(c_int) :: apiVersion
  end type VkApplicationInfo

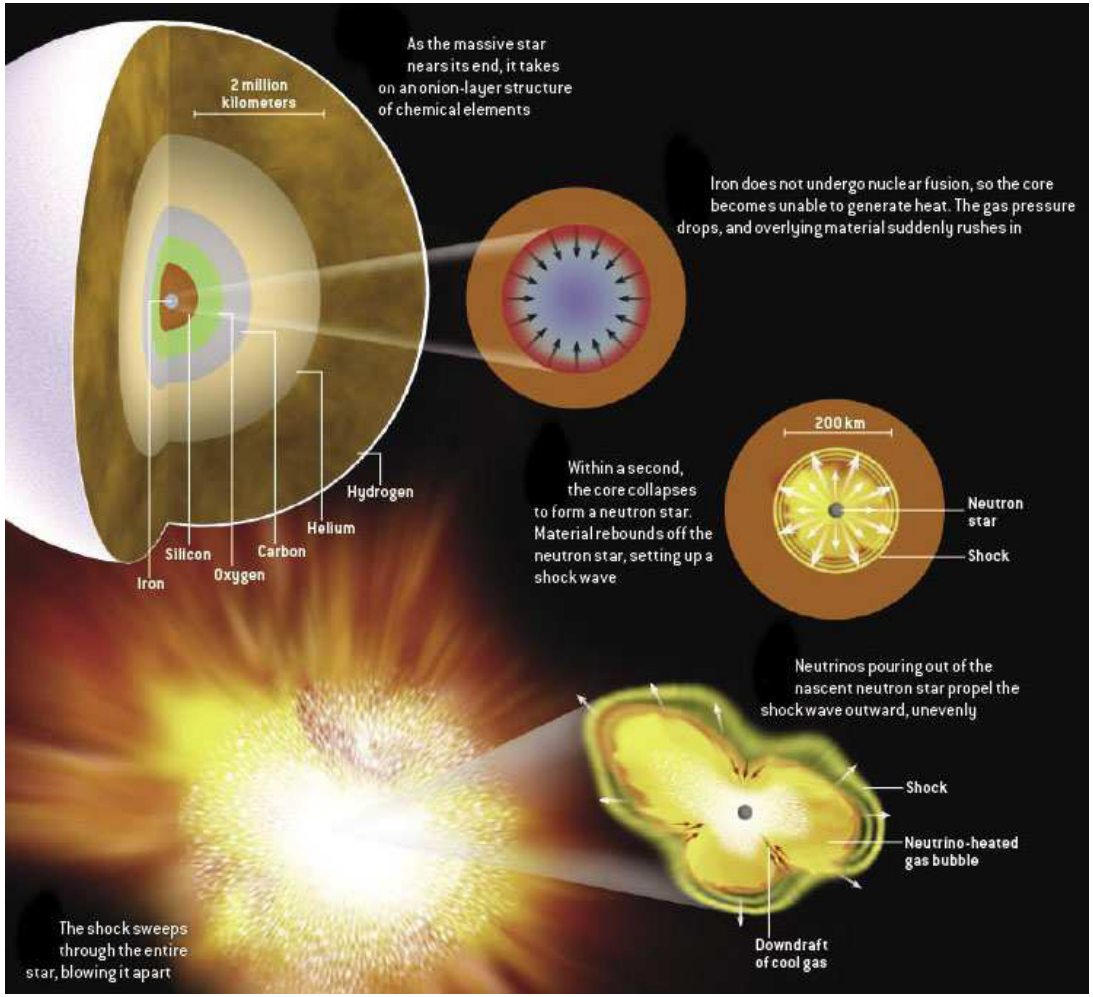
  type, bind(c) :: VkInstanceCreateInfo
    integer(c_int) :: sType
    type(c_ptr) :: pApplicationInfo
    integer(c_int) :: enabledLayerCount
    type(c_ptr) :: ppEnabledLayerNames
    integer(c_int) :: enabledExtensionCount
    type(c_ptr) :: ppEnabledExtensionNames
  end type VkInstanceCreateInfo

  type(c_ptr), bind(c, name="VK_NULL_HANDLE") :: VK_NULL_HANDLE
```

Capture d'électrons & supernovae

Groupe théorie IPNO
Simulation :
1000 types de noyaux
50 valeurs de T
$\Rightarrow$ 1000 ans de CPU

* *profiling*
* audit bibliothèques
* gestion mémoire
* vectorisation



$\Rightarrow$ **facteur $>$ 55**

$\Rightarrow$ **20 ans de CPU après accélération**

Structure du programme

```
(... loops on J, Π ...)                ! 6
(... nuclear structure computation following RPA model: ...)
(... energy level occupations, wavefunctions wf / dwf ...)
do idxenergy0 = 0, nbenergystep - 1    ! 150 electron energy level
   do energyc = 1, nconf               ! 153 / 405 / 540 kernel energy level
      do k_simps = 0, n_simps          ! 31 electron-neutrino angles
         do pairc = 1, nconf           ! 153 / 405 / 540 kernel energy level T < 1 MeV
                                       ! 182 / 476 / 637 kernel energy level T = 2 MeV
                                       ! 232 / 609 / 821 kernel energy level T = 3 MeV
                                       ! 386 / 1041 / 1448 kernel energy level T = 4 MeV
            do i = 1, maxr             ! 168 radial wavefunctions mesh
               … bottom-most loop: spherical bessel functions:> 92% of CPU
```

$$j_\ell(qr)$$

nid de 5 boucles indépendantes

$\Longrightarrow$ paradis des parallélistes

entre 121 millions et 11 milliards d'itérations seulement pour les 4 boucles internes

Parallelisation ?

La parallelisation a l'air facile ⇒ OpenMP

```
(... loops on J, Π ...)                ! 6
!$OMP PARALLEL PRIVATE (Eelectron, ecsum) SHARED (Energy_electron, sigma)
!$    & COPYIN (/energyidx/, /rpamix/, /bwf/, /occupa/, /qval/, /bdiam/,
!$    & /bqwf/, /bwu2/, /bnri/, /bnrir/, /bwu1/, /bwu1r/, /blecp1/,
!$    & /blecp2/, /bptl/)
do idxenergy0 = 0, nbenergystep - 1    ! 150 electron energy level
   do energyc = 1, nconf               ! 153 / 405 / 540 kernel energy level
      do k_simps = 0, n_simps          ! 31 electron-neutrino angles
         do pairc = 1, nconf           ! 153 / 405 / 540 kernel energy level
            do i = 1, maxr             ! 168 radial wavefunctions mesh
               … bottom-most loop: spherical bessel functions:> 92% of CPU
```

⇒ segmentation fault :(

IDRIS ⇒ *convertissez tout dans un seul langage*

Fortran 90, validation des résultats, ajout de OpenMP

⇒segmentation fault :(

* Faut-il simplifier le code ?
* Dur avec des threads, facile avec des process
* Threads pas si équivalents en durée...
* Transformer le code pour exprimer une transformée rapide de Bessel Sphérique *à la* FFT ?

Vectorization

```
gfortran -Ofast -march=native -mtune=native -mavx2
    -fopenmp -fopenmp-simd -ftree-vectorize -fopt-info
       -ftree-vectorizer-verbose=1
    -mrecip --param=ssp-buffer-size=4
    -g -Wall -Wextra -Wshadow -Wconversion -Wpedantic
    speedcbrtf.f90 -o speedcbrtf -lm && \
taskset --cpu-list 0 ./speedcbrtf
```

Vectorization

```
module cbrt_module
  use, intrinsic :: iso_c_binding, only: c_float
  use, intrinsic :: iso_fortran_env, only: real32, int32
  use, intrinsic :: ieee_arithmetic

contains

  elemental real (real32) function naive_cbrt (a)
    real (real32), intent (in) :: a
    naive_cbrt = exp (log (a) / 3)
  end function naive_cbrt

  !$omp declare simd
  elemental function nw_cbrt (a) result (r)
    use, intrinsic :: ieee_arithmetic
    real (real32), intent (in) :: a
    real (real32) :: r
    real (real32) :: b, u, v
    real (real32) :: bb, uu, vv
    integer                         :: e, f, s
    real (real32), parameter, dimension (*) :: c = [ &
         real (z'3f179960', real32), &
         real (z'3fb16615', real32), &
         real (z'3fbaa370', real32), &
         real (z'3eae82c8', real32) ]

    b = abs (a) ! strip off sign-bit
    ! Extract exponent, then adjust    ! e = exponent (b)
    e = ibits(transfer (b, 1_int32), 23, 8) - 126

    ! Extract the mantissa    ! b = fraction (b)
    b = transfer (ior (ibits(transfer (b, 1_int32), 0, 23), z'3F000000'), 1.0_real32)
```

OpenACC

```
subroutine radproint_array (j0)
  implicit none
  integer, intent (in) :: j0
  integer :: k_simps, pairc, idxenergy, energyc, maj ! dummy indices
  integer :: Jmin, Jmax
  Jmin = max (j0-1, 0)
  Jmax = j0+1

  allocate (Rad_proint1_array  (1:nconf, 0:n_simps, 1:nconf, 0:nbenergystep-1, Jmin:Jmax))

  !$acc data present_or_create (mask_kinematics, Rad_proint2B_array, Rad_proint2A_array, Rad_pro
  !$acc                        wf12_array, Bess_f_array, wf12inv_array, wf1dwf2_array)

  !$acc parallel loop collapse (3)
  build10: do maj = Jmin, Jmax
    build11: do idxEnergy = 0, nbenergystep-1
      build12: do energyc = 1, nconf
        if (mask_kinematics (energyc, idxEnergy)) then
  !$acc loop vector independent collapse (2)
          build13: do k_simps = 0, n_simps
            build14: do pairc = 1, nconf
              Rad_proint1_array (pairc, k_simps, energyc, idxEnergy, maj) = &
              sum (wf1dwf2_array (1:maxr, pairc) * Bess_f_array (1:maxr, k_simps, energyc, idxEn
            end do build14
          end do build13
        end if
      end do build12
    end do build11
  end do build10
  !$acc end parallel loop

  !$acc parallel
```

CUDAFortran

```
!  nvfortran -g -O4 -fast testSaxpy.cuf -o testSaxpy
module mathOps
contains
  attributes(global) subroutine saxpy(x, y, a)
    implicit none
    real :: x(:), y(:)
    real, value :: a
    integer :: i, n
    n = size(x)
    i = blockDim%x * (blockIdx%x - 1) + threadIdx%x
    if (i <= n) y(i) = y(i) + a*x(i)
  end subroutine saxpy
end module mathOps

program testSaxpy
  use mathOps
  use cudafor
  implicit none
  integer, parameter :: N = 40000
  real :: x(N), y(N), a
  real, device :: x_d(N), y_d(N)
  type(dim3) :: grid, tBlock

  tBlock = dim3(256,1,1)
  grid = dim3(ceiling(real(N)/tBlock%x),1,1)

  x = 1.0; y = 2.0; a = 2.0
  x_d = x
  y_d = y
  call saxpy<<<grid, tBlock>>>(x_d, y_d, a)
  y = y_d
  write(*,*) 'Max error: ', maxval(abs(y-4.0))
```

Réaction de diffusion de Gray Scott

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto

On mélange deux réactifs et on produit un troisième (espèces U, V et P) :

$$\begin{aligned} U + 2V &\rightarrow 3V \\ V &\rightarrow P \end{aligned}$$

$\Longrightarrow$ **Autocatalytique** : V transforme U en lui-même quand il est suffisamment concentré, puis V se transforme en P qui ne réagit plus (il existe de telles molécules).

$$\begin{aligned} \frac{\partial u}{\partial t} &= r_u \nabla^2 u - uv^2 + f_r(1-u) \\ \frac{\partial v}{\partial t} &= r_v \nabla^2 v + uv^2 - (f_r + k_r)v \end{aligned}$$

- u et v, concentration des espèces U et V
- r_u et r_v, taux de diffusion de U et V
- k_r (Kill Rate), taux de conversion de V en P
- f_r (Feed Rate), vitesse du processus qui nourrit U et tue V et P
- $\nabla^2 u$ et $\nabla^2 v$, différence locale de concentration spatiale entre la cellule et ses voisines.

Build

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto

```
#!/usr/bin/make
SHELL = /bin/sh
MAIN ?= base_gray_scott_from_make_90

# FC = the Fortran compiler to use
FC ?= gfortran

ifeq ($(FC), ifx)
  MAIN:=$(MAIN)_ifx
  OBJ = objintel
  FFLAGS = -g -traceback -check bounds -warn all -Wextra -Wpedantic -Wconversion -Wshadow
  FFLAGS= -Ofast -vec -xHost -axCORE-AVX2 -mavx2 -align array64byte -ipo -ffast-math -fp-model fas
else ifeq ($(FC), nvfortran)
  MAIN:=$(MAIN)_nv
  OBJ = objnv
  FFLAGS = -g -Wall -Wextra -Werror
  FFLAGS+= -tp=haswell -C -march=native -mtune=native -mavx2 -fopenmp -stdpar -Minfo=accel # -gpu=
#  FFLAGS+= -tp=haswell -C -march=native -mtune=native -mavx2 -fopenmp -stdpar=multicore -Minfo #-
else # ifeq ($(FC), gfortran)
  MAIN:=$(MAIN)_gnu
  OBJ = obj
  FFLAGS = -g -Wall -Wextra -Wpedantic -Wconversion -Wshadow -Werror -fimplicit-none -finit-real=s
  FFLAGS+= -Ofast -march=native -mtune=native -mavx2 -ffast-math
  FFLAGS+= -malign-double -mcmodel=medium -funroll-loops -ftree-vectorize -fopt-info -ftree-vector
  FFLAGS+= -fopenmp -fopenmp-simd
```

Build

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto

```
ARCHI := $(shell LC_ALL=C uname -m)
HDF_INCLUDES   = /usr/include/hdf5/serial # package libhdf5-dev
HDF_LIB   = /usr/lib/$(ARCHI)-linux-gnu/hdf5/serial # package libhdf5-dev libhdf5-103… / libhdf5-1

PROJECT_DATE := $(shell LC_ALL=C date +%Y%m%d_%H%M%S)
#DEFINES       = -DVERSION=\'$(shell LC_ALL=C hg id -n):$(shell LC_ALL=C hg id -i)\' -DLASTDATE=\'
#DEFINES       = -DVERSION=\'$(shell LC_ALL=C hg tip --template '{parent}'|sed -e 's/ //')\' -DLAS
DEFINES       = -DVERSION=\'$(shell LC_ALL=C git describe --abbrev=12 --always --dirty=+)\' -DLAST
CPPFLAGS      = $(DEFINES) -DCPP_VERSION=\'$(CPP_VERSION)\'
CPPFLAGS     += -I$(HDF_INCLUDES)
```

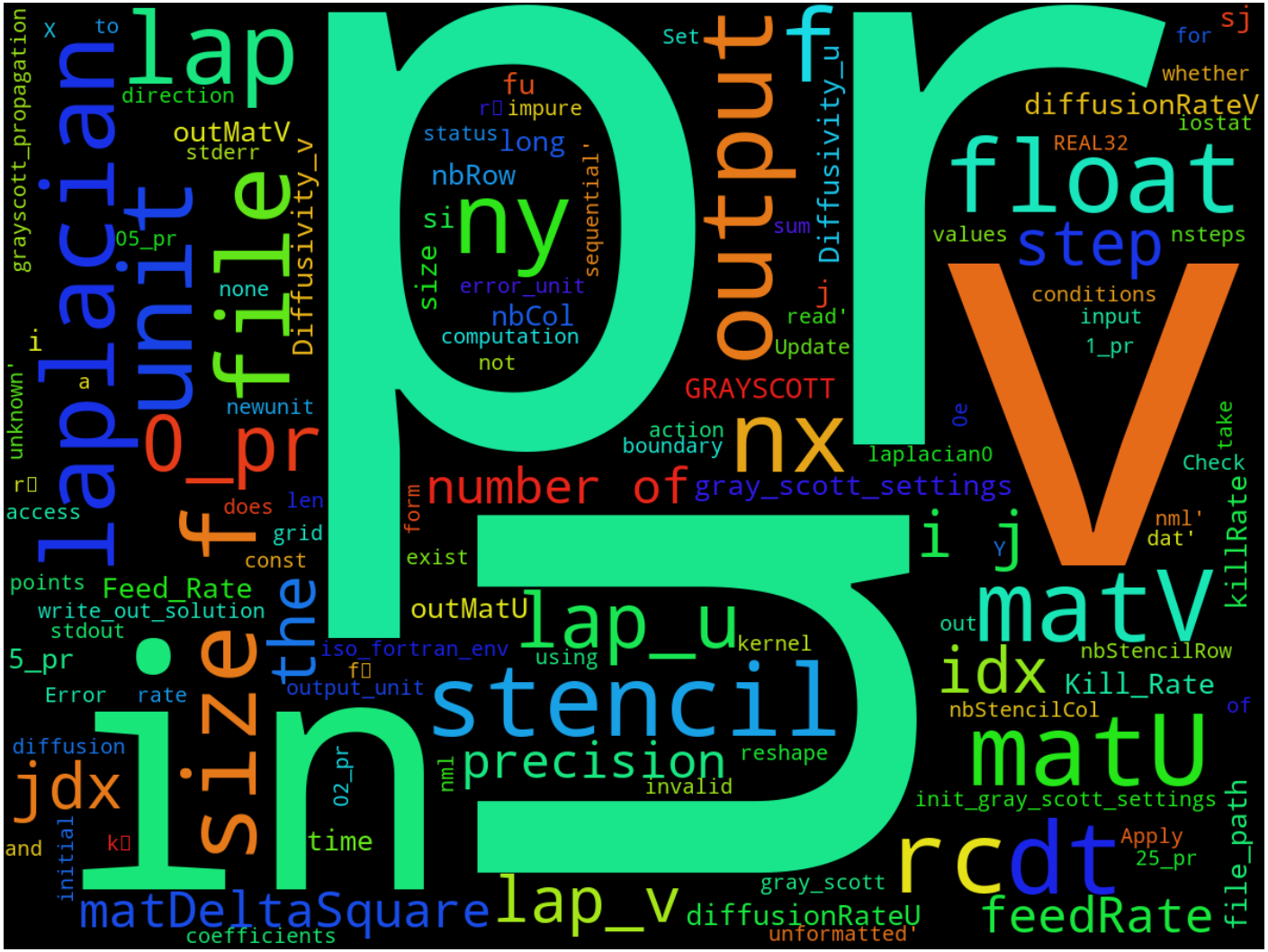
Build

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto

```
lafage@ll-lafage:~/GrayScott$ make help
# target: base_gray_scott_from_make_90 - build main executable (default target)
# target: help - Display callable targets.
# target: clean - clean all intermediate build files
# target: clobber - clean all intermediate build files, executable and coredump
# target: display - timed execution of main executable applied on data
# target: run - timed execution of main executable applied on data
# target: perf - perf evaluation on monocore execution of main executable applied on data
# target: maqao - vectorization analysis of main executable
# target: memcheck - memchecked execution of main executable applied on data
# target: callgrind - profiled execution of main executable applied on data
# target: cachegrind - profiled execution of main executable applied on data
# target: doc - Doxygen documentation from literate source code.
# target: check - static check of Fortran source code (linter).
# target: camfort - static check of source code (linter).
# target: simian - static check of duplicate source code (linter).
# target: lizard - Cyclomatic Complexity Analyzer of source code (linter).
# target: fortitude - Modern Fortran linter.
# target: cloc - another SLOC count of source code.
# target: sloccount - SLOC count of Fortran source code.
# target: wordcloud - display wordcloud of Fortran source code.
# target: validate_typo - check number of typo mistakes ridden files, by category.
```

wordcloud

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto



Build

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto

```
LDFLAGS=$(FFLAGS)
LDLIBS=-L$(HDF_LIB) -lhdf5

SOURCES=gray_scott.f90
PRJS= $(SOURCES:.f=.prj)

# target: base_gray_scott_from_make_90 - build main executable (default target)
$(MAIN): $(SOURCES) $(OBJ)
	cd $(OBJ) ; $(FC) $(LDFLAGS) $(CPPFLAGS) ../$< -o ../$@ $(LDLIBS) ; cd ..

$(OBJ):
	mkdir -p $(OBJ)

# target: help - Display callable targets.
help:
	@egrep "^# target:" $(MAKEFILE_LIST)

# target: clean - clean all intermediate build files
clean:
	$(RM) -f *.o *.mod

# target: clobber - clean all intermediate build files, executable and coredump
clobber:
	$(RM) *.o *.mod core $(MAIN)

# target: display - timed execution of main executable applied on data
display: $(MAIN)
	strings ./$<|grep -E '(build_date|revision_date|revision|compiler_version| version):'
```

Source

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto

```
module precision
  use, intrinsic :: iso_fortran_env, only: pr => REAL32
end module precision

module gray_scott_settings
  use precision
  integer ::                          &
       nx =  960,                     & ! number of grid points in X direction
       ny =  540,                     & ! number of grid points in Y direction
   nsteps = 340                         ! number of time steps to take
  integer, parameter ::               &
       intersave_count = 34,          & !
       input  = 10,                   & ! logical unit number of output file
       output = 11                      ! logical unit number of output file
  real (pr) ::                        &
       dt = 0.1_pr,                   & ! time step
       Diffusivity_u = 2.0e-5_pr,     & ! r_u: diffusion rate of u
       Diffusivity_v = 1.0e-5_pr,     & ! r_v: diffusion rate of v
       Feed_Rate = 0.02_pr,           & ! f_r
       Kill_Rate = 0.05_pr              ! k_r
  real (pr), allocatable, dimension (:, :) :: u, v, Unew, Vnew
  real (pr), allocatable, dimension (:, :) :: lap_u, lap_v

  real (pr), dimension (-1:1, -1:1), parameter ::  & ! Stencil coefficients
       stencil = reshape ([ real (pr) :: 0 , 1, 0, &
                                         1, -4, 1, &
                                         0,  1, 0], [3, 3])
```

Source

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto

```
  subroutine init_gray_scott_settings
    use, intrinsic :: iso_fortran_env, only: stderr => error_unit, stdout => output_unit
    character (len=*),  parameter     :: file_path = 'gray_scott_settings.nml'
    integer                           :: fu, rc
    ! Check whether file exists.
    inquire (file=file_path, iostat=rc)
    if (rc /= 0) then
       write (stderr, '("Error: input file ", a, " does not exist")') file_path
       return
    end if
    ! Open and read Namelist file.
    open (action='read', file=file_path, iostat=rc, newunit=fu)
    read (nml=GRAYSCOTT, iostat=rc, unit=fu)
    if (rc /= 0) write (stderr, '("Error: invalid Namelist format")')
    write (nml=GRAYSCOTT, unit=stdout)
    write (nml=GRAYSCOTT_SIZE, unit=stdout)
    close (unit=fu)
    if (.not. allocated (u))     allocate (u (nx, ny))
    if (.not. allocated (v))     allocate (v (nx, ny))
    if (.not. allocated (unew))  allocate (unew (nx, ny))
    if (.not. allocated (vnew))  allocate (vnew (nx, ny))
    if (.not. allocated (lap_u)) allocate (lap_u (nx, ny))
    if (.not. allocated (lap_v)) allocate (lap_v (nx, ny))
  end subroutine init_gray_scott_settings
```

Source

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto

```
    ! Apply the stencil kernel for the laplacian computation
    ! $ omp do private(i, j)
!    do j= 2,size (f, 2) - 1
!       do i= 2,size (f, 1) - 1
    do concurrent (j= 2:size (f, 2) - 1, i= 2:size (f, 1) - 1)
          lap (i, j) = sum (stencil * f (i-1:i+1, j-1:j+1))
!       end do
    end do
    ! $ omp end do
  end function laplacian

  pure subroutine init_fields (u, v)
    real (pr), dimension (:, :), intent (inout) :: u, v
    ! Set initial conditions
    u = 1.0_pr
    v = 0.0_pr
    u (nx/2:nx/2 + 15, ny/2:ny/2 + 15) = 0.5_pr
    v (nx/2:nx/2 + 15, ny/2:ny/2 + 15) = 0.25_pr

    ! Set boundary conditions
    ! u (:, 1) = 0.5_pr
    ! v (:, 1) = 0.25_pr
  end subroutine init_fields
```

Configuration

https://gitlab.in2p3.fr/lafage/GrayScottFortranTuto

gray_scott_settings.nml:

```
&GRAYSCOTT
 DT= 1.0,                  ! time step
 DIFFUSIVITY_U=  0.1,      ! r_u: diffusion rate of u
 DIFFUSIVITY_V=  0.05,     ! r_v: diffusion rate of v
 FEED_RATE=  0.014,        ! f_r
 KILL_RATE=  0.054,        ! k_r
 /
```

gray_scott.f90:

```
  namelist /GRAYSCOTT/ dt, Diffusivity_u, Diffusivity_v, Feed_Rate, Kill_Rate
  namelist /GRAYSCOTT_SIZE/ nx, ny, nsteps
```

- nvfortran
  les dernières nouveautés de la Norme sont implémentées de manières variable)
- quand on a bien exprimé le parallélisme
  l'expression peut déborder de la mémoire GPU)
- ⇒ obstruction à la vocation de Fortran
- faut-il des prépilateurs ?

BILAN : c'est chouette, mais...