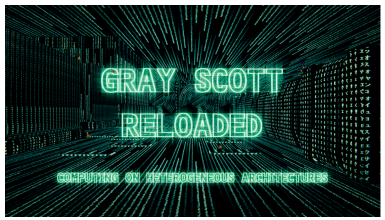
Vectorization : Data parallelism

Row & Order

Special Vectorization Unit

Vincent Lafage

[1]IJCLab, Laboratoire de Physique des 2 Infinis Irène Joliot-Curie
Université Paris-Saclay



July 1st 2024

Scales of parallelism

- Processes (in well separated environments) :
  - so called *embarassingly parallel* problems
    — particle physics event-wise distributed processing on a grid
  - inter-process communication *à la Unix*
    — pipe
    — shared memory
    — semaphore
  - with separate computer and message passing across the network (distributed memory systems) :
    — `PVM` *« Parallel Virtual Machine »* 1989
    — `MPI` *« Message Passing Interface »* 1994

- Threads (*a.k.a. « lightweight processes »*)
  - Reduced Overhead : *compared to a process*
    — `C/C++ pthread` *« POSIX thread »* 1995
    — `Ada task` / protected objects
  - Shared Memory on SMP *« Symmetric Multi Processing »* machines :
    — `OpenMP` *« Open Multi Processing »* 1997.

- Vectorization :
  - even thread have too much overhead when concurrent tasks are elementary :
    ⇒ we can use no CPU thread for individual sum or products of many data
  - ILP *« Instruction Level Parallelism »*
    — vector processor *« à la Cray »* : SIMD *« Single Instruction, Multiple Data »*
    — modern processor with fixed length SIMD

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

How to vectorize ? ordering data

We know *Bit Level Parallelism* :
we can AND or OR 8, 16, 32, 64 bits in one instruction
In the same way, for our vector instruction to perform fluently, we need to align our vector of data on memory boundary.

We have better order data of same nature as an (aligned) (contiguous) array
$\Rightarrow$ for data contiguity we prefer a

structure of array

to the classical object-oriented

collection (array) of objects (structure)

How to vectorize ?

Vectorization, an industrialization process :

see your computation as a production line

Pincer movement around performance :

- parallelizing from the top,
- vectorizing from the bottom

... + keep the cache hot !

How to vectorize ?

1. handcraft vector code with dedicated low level instructions (intrinsics)
2. use vector libraries
3. let the compiler do it, provided we can
   1. express the data parallelism (high-level aspect)
      $\Rightarrow$ Harnessing the Power of Arrays (*à la Matlab / Fortran 90*)
   2. hint the data contiguity & alignement to the compiler (low-level aspect)

And then, how to check that it was properly vectorized ?

- compiler's flagS to report vectorisation
- check the resulting assembly (yuk !)
- `maqao`
- ...indirectly, `perf`

How to vectorize ?

Complex numbers : a symbolic vectorization
... that gets in the way of practical vectorization !

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

We need a complex of vector rather than a vector of complex.

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