Low and High Performance Computing in HEP

Michael N. Borinsky
Supervisor: Lorenzo Moneta
PH - SFT

Humboldt-University Berlin

15. August 2012

Why Low or High Performance Computing?

Why Low or High Performance Computing?

Answer:

To program and to run code costs time and money!

Why Low or High Performance Computing?

Answer:

To program and to run code costs time and money!

The lifetime of an ordinary physics application consists of:

Why Low or High Performance Computing?

Answer:

To program and to run code costs time and money!

The lifetime of an ordinary physics application consists of:

Time it takes to write the code:

$$T_{develop}$$

Why Low or High Performance Computing?

Answer:

To program and to run code costs time and money!

The lifetime of an ordinary physics application consists of:

Time it takes to write the code:

$T_{develop}$

Time the code is used:

$T_{running}$

Low Performance Computing

Applicable if:

$$T_{\text{develop}} \gg T_{running}$$

Low Performance Computing

Applicable if:

$$T_{\text{develop}} \gg T_{running}$$

- High-Level programming languages should be used
  e.g. python, mathematica, gnuplot.

Low Performance Computing

Applicable if:

$$T_{\text{develop}} \gg T_{running}$$

- High-Level programming languages should be used
  e.g. python, mathematica, gnuplot.
- Rapid programming is applicable
  i.e. write "Quick and Dirty" code.

Low Performance Computing

Applicable if:

$$T_{\text{develop}} \gg T_{\textit{running}}$$

- High-Level programming languages should be used
  e.g. python, mathematica, gnuplot.
- Rapid programming is applicable
  i.e. write "Quick and Dirty" code.
- No time should be spend on optimization!

Low Performance Computing

Applicable if:

$$T_{\text{develop}} \gg T_{\textit{running}}$$

- High-Level programming languages should be used
  e.g. python, mathematica, gnuplot.
- Rapid programming is applicable
  i.e. write "Quick and Dirty" code.
- No time should be spend on optimization!
- First priority is to get the code running.

High Performance Computing

Applicable if:

$$T_{\text{running}} \gg T_{develop}$$

High Performance Computing

Applicable if:

$$T_{\text{running}} \gg T_{develop}$$

- Low-Level programming languages can be used
  e.g. C/C++, FORM, ROOT.

High Performance Computing

Applicable if:

$$T_{\text{running}} \gg T_{develop}$$

- Low-Level programming languages can be used
  e.g. C/C++, FORM, ROOT.
- Elaborate planning and testing is obligatory! Bugs can be fatal...

High Performance Computing

Applicable if:

$$T_{\text{running}} \gg T_{develop}$$

- Low-Level programming languages can be used
  e.g. C/C++, FORM, ROOT.
- Elaborate planning and testing is obligatory! Bugs can be fatal...
- Optimization is applicable if T_{running} is large and costly.

High Performance Computing

Applicable if:

$$T_{\text{running}} \gg T_{develop}$$

- Low-Level programming languages can be used
  e.g. C/C++, FORM, ROOT.
- Elaborate planning and testing is obligatory! Bugs can be fatal...
- Optimization is applicable if T_{running} is large and costly.
- The computing architecture needs to be taken into account.

Computing Architecture

Computing Architecture

The status quo is Cluster Computing!

Computing Architecture

The status quo is Cluster Computing!

- Needs to be bought and maintained or rent.
  $\Rightarrow$ expensive!

Computing Architecture

The status quo is Cluster Computing!

- Needs to be bought and maintained or rent.
  $\Rightarrow$ expensive!

Two basic properties of a typical cluster:

Computing Architecture

The status quo is Cluster Computing!

- Needs to be bought and maintained or rent.
  $\Rightarrow$ expensive!

Two basic properties of a typical cluster:

- The speed/bandwidth of
  machines
  High for modern GPUs

Computing Architecture

The status quo is Cluster Computing!

- Needs to be bought and maintained or rent.
  $\Rightarrow$ expensive!

Two basic properties of a typical cluster:

- The speed/bandwidth of machines
  High for modern GPUs

$\Rightarrow$ Code Vectorization

Computing Architecture

The status quo is Cluster Computing!

- Needs to be bought and maintained or rent.
  $\Rightarrow$ expensive!

Two basic properties of a typical cluster:

- The speed/bandwidth of machines
  High for modern GPUs
- The total number of computers or processors in the cluster.

$\Rightarrow$ Code Vectorization

Computing Architecture

The status quo is Cluster Computing!

- Needs to be bought and maintained or rent.
  $\Rightarrow$ expensive!

Two basic properties of a typical cluster:

- The speed/bandwidth of machines
  High for modern GPUs
- The total number of computers or processors in the cluster.

$\Rightarrow$ Code Vectorization

$\Rightarrow$ Code Parallelization

Computing Architecture

The status quo is Cluster Computing!

- Needs to be bought and maintained or rent.
  $\Rightarrow$ expensive!

Two basic properties of a typical cluster:

- The speed/bandwidth of machines
  High for modern GPUs
- The total number of computers or processors in the cluster.

$\Rightarrow$ Code Vectorization

$\Rightarrow$ Code Parallelization

Both are non trivial!

Vectorization

Vectorization

CPU clock rates are not increasing much anymore.

Newer CPUs can process whole chunks of data at once instead.

Vectorization

CPU clock rates are not increasing much anymore.
Newer CPUs can process whole chunks of data at once instead.

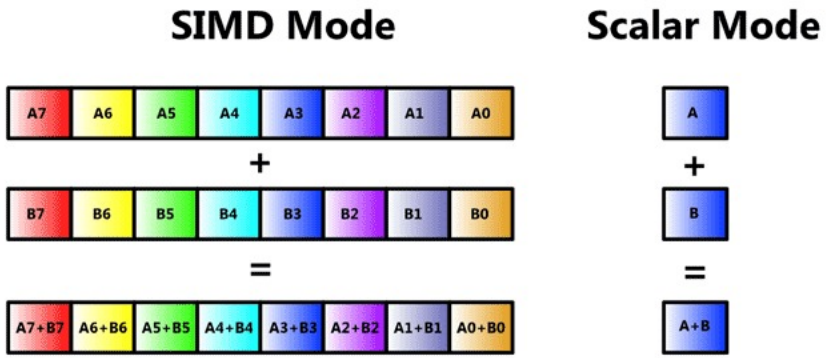

Figure: Illustration of a vector architecture

Vectorization

CPU clock rates are not increasing much anymore.
Newer CPUs can process whole chunks of data at once instead.

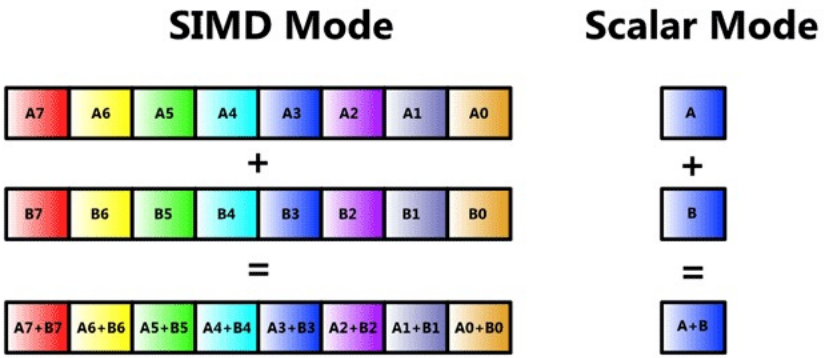

Figure: Illustration of a vector architecture

For your code to benefit you need to:

Vectorization

CPU clock rates are not increasing much anymore.
Newer CPUs can process whole chunks of data at once instead.

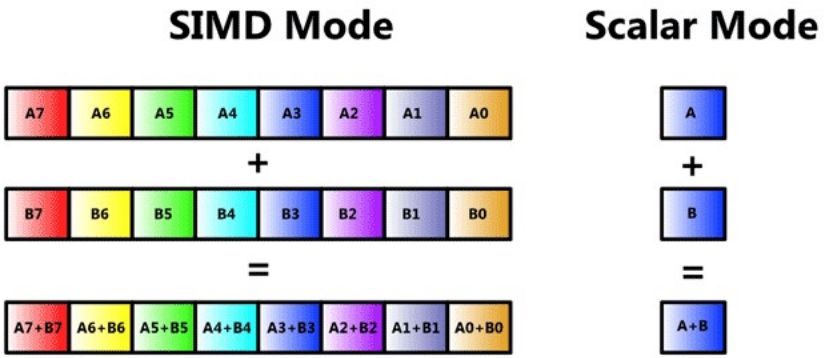

Figure: Illustration of a vector architecture

For your code to benefit you need to:

- Deal with consecutive chunks of data.

Vectorization

CPU clock rates are not increasing much anymore.
Newer CPUs can process whole chunks of data at once instead.

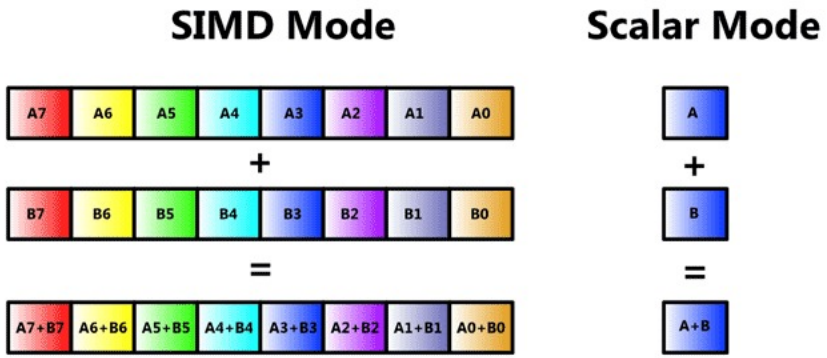

Figure: Illustration of a vector architecture

For your code to benefit you need to:

- Deal with consecutive chunks of data.
- Store the data consecutively in memory.

Vectorization

CPU clock rates are not increasing much anymore.
Newer CPUs can process whole chunks of data at once instead.

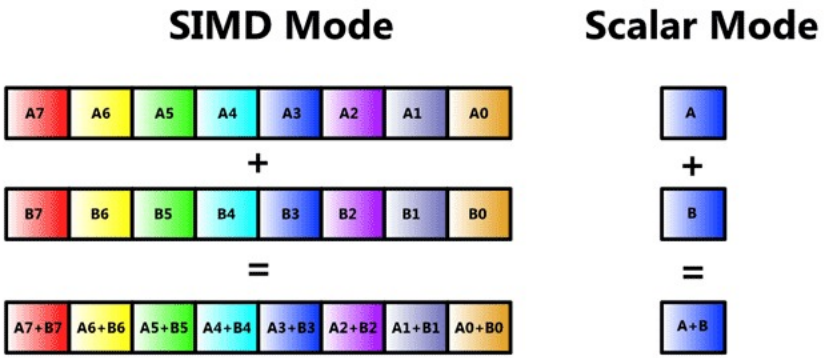

Figure: Illustration of a vector architecture

For your code to benefit you need to:

- Deal with consecutive chunks of data.
- Store the data consecutively in memory.
- Plan ahead! Already written code is hard to vectorize.

Parallelization

Parallelization

It is hard to change the architecture of an existing cluster.
But adding more CPUs to the cluster is easy and cheap.

Parallelization

It is hard to change the architecture of an existing cluster.
But adding more CPUs to the cluster is easy and cheap.
For your code to benefit you need to:

Parallelization

It is hard to change the architecture of an existing cluster.
But adding more CPUs to the cluster is easy and cheap.
For your code to benefit you need to:

- Split the task into small sub tasks.

Parallelization

It is hard to change the architecture of an existing cluster.
But adding more CPUs to the cluster is easy and cheap.
For your code to benefit you need to:

- Split the task into small sub tasks.
- Take care there are no dependencies
  e.g. no global variables.

Parallelization

It is hard to change the architecture of an existing cluster.
But adding more CPUs to the cluster is easy and cheap.
For your code to benefit you need to:

- Split the task into small sub tasks.
- Take care there are no dependencies
  e.g. no global variables.
- Plan ahead! Already written code is hard to parallelize.

Example: Parallelization and Vectorization in ROOT

Example: Parallelization and Vectorization in ROOT

Reminder: Consider the formula for the χ^2 value

$$\chi^2 = \sum_i \frac{(y_i - f(x_i))^2}{\sigma_i^2}$$

Example: Parallelization and Vectorization in ROOT

Reminder: Consider the formula for the χ^2 value

$$\chi^2 = \sum_i \frac{(y_i - f(x_i))^2}{\sigma_i^2}$$

The sum on the right hand side can be vectorized.

Example: Parallelization and Vectorization in ROOT

Reminder: Consider the formula for the χ^2 value

$$\chi^2 = \sum_i \frac{(y_i - f(x_i))^2}{\sigma_i^2}$$

The sum on the right hand side can be vectorized.

Performing a fit, χ^2 will be a function of some parameters:

$$\chi^2(a, b, ...)$$

Example: Parallelization and Vectorization in ROOT

Reminder: Consider the formula for the χ^2 value

$$\chi^2 = \sum_i \frac{(y_i - f(x_i))^2}{\sigma_i^2}$$

The sum on the right hand side can be vectorized.

Performing a fit, χ^2 will be a function of some parameters:

$$\chi^2(a, b, ...)$$

$\chi^2(a, b, ...)$ needs to be minimized.
$\Rightarrow$ Evaluations at many different points.

Example: Parallelization and Vectorization in ROOT

Reminder: Consider the formula for the χ^2 value

$$\chi^2 = \sum_i \frac{(y_i - f(x_i))^2}{\sigma_i^2}$$

The sum on the right hand side can be vectorized.

Performing a fit, χ^2 will be a function of some parameters:

$$\chi^2(a, b, ...)$$

$\chi^2(a, b, ...)$ needs to be minimized.
$\Rightarrow$ Evaluations at many different points.

This can be parallelized, because different points in parameter space are independent.

Conclusions

Conclusions

If you come to a point, where you need computation:

Conclusions

If you come to a point, where you need computation:

- Choose between low or high performance computing.

Conclusions

If you come to a point, where you need computation:

- Choose between low or high performance computing.
- In doubt, choose low performance computing!

Conclusions

If you come to a point, where you need computation:

- Choose between low or high performance computing.
- In doubt, choose low performance computing!
- Choose the appropriate tools.

Conclusions

If you come to a point, where you need computation:

- Choose between low or high performance computing.
- In doubt, choose low performance computing!
- Choose the appropriate tools.

In case of high performance computing:

Conclusions

If you come to a point, where you need computation:

- Choose between low or high performance computing.
- In doubt, choose low performance computing!
- Choose the appropriate tools.

In case of high performance computing:

- Plan ahead and test your code.

Conclusions

If you come to a point, where you need computation:

- Choose between low or high performance computing.
- In doubt, choose low performance computing!
- Choose the appropriate tools.

In case of high performance computing:

- Plan ahead and test your code.
- Use sophisticated techniques to minimize costs/time demands.