Computational Plasma Physics

Summer University for Plasma Physics, September 2016

M. Drevlak, Max-Planck-Institut für Plasmaphysik, Teilinstitut Greifswald

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053.
The views and opinions expressed herein do not necessarily reflect those of the European Commission.

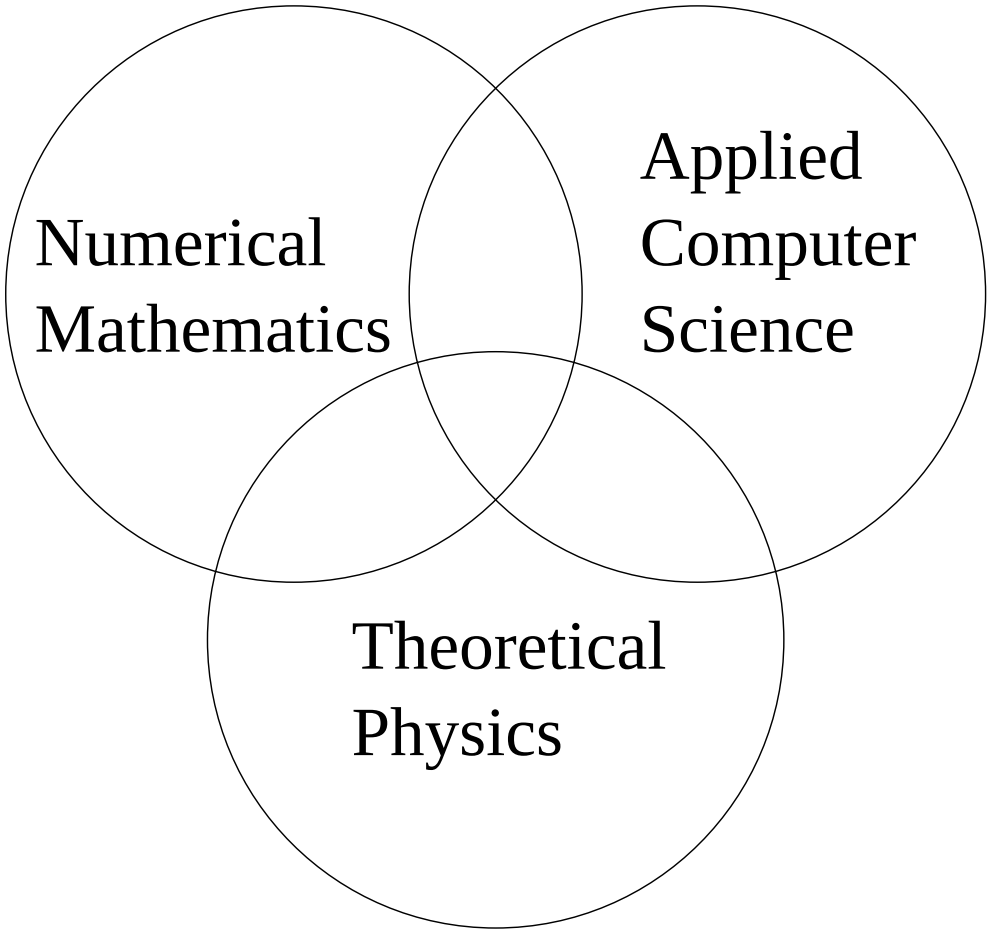
- Computational physics - why bother?
- Basic techniques - not exhaustive
- Numerical libraries - a few should do
- Computers - adapt to architecture
- Parallelisation - everywhere
- Example codes - tools for a lifetime

Computational Plasma Physics - Why Bother?

- High dimensionality
  Example: kinetic effects in systems off thermal equilibrium
- Non-linearity
  Example: MHD theory
- Input data/initial values
  Example: analysis of experimental data
- Complex Geometry
  Example: EM field or any other PDE
- Complex systems
  Example: plasma edge
- Non-local effects
  Example: magnetic field lines
- (Almost) nothing left to solve analytically
  Sorry - no examples!

The Nature of Computational Physics

- located at the intersection between different disciplines
- provides understanding of experimental results resisting analytical investigation
- tries to make predictions about future experiments
- leads to design of new devices

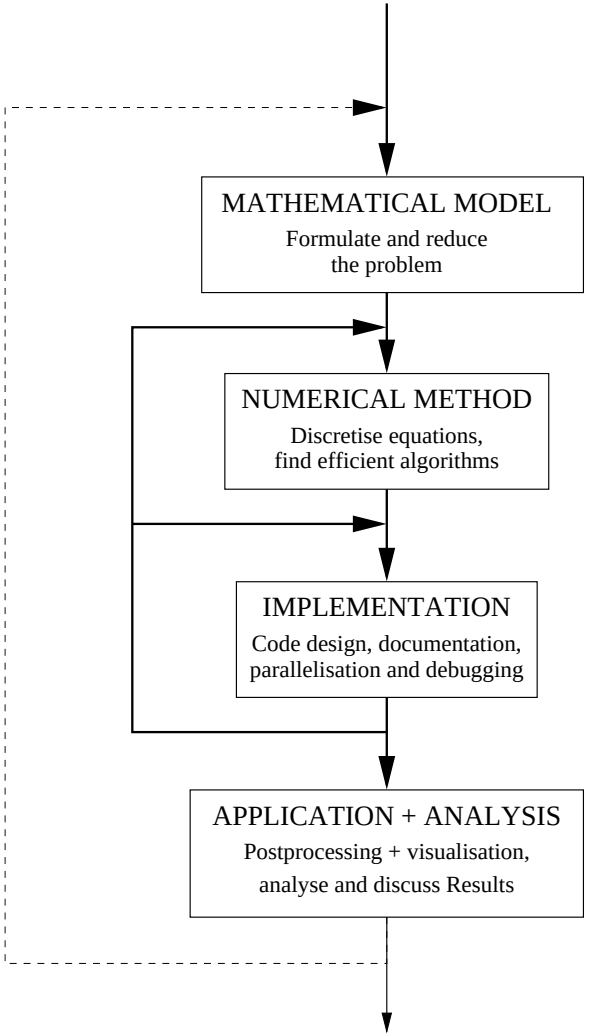


The Workflow in Computational Physics

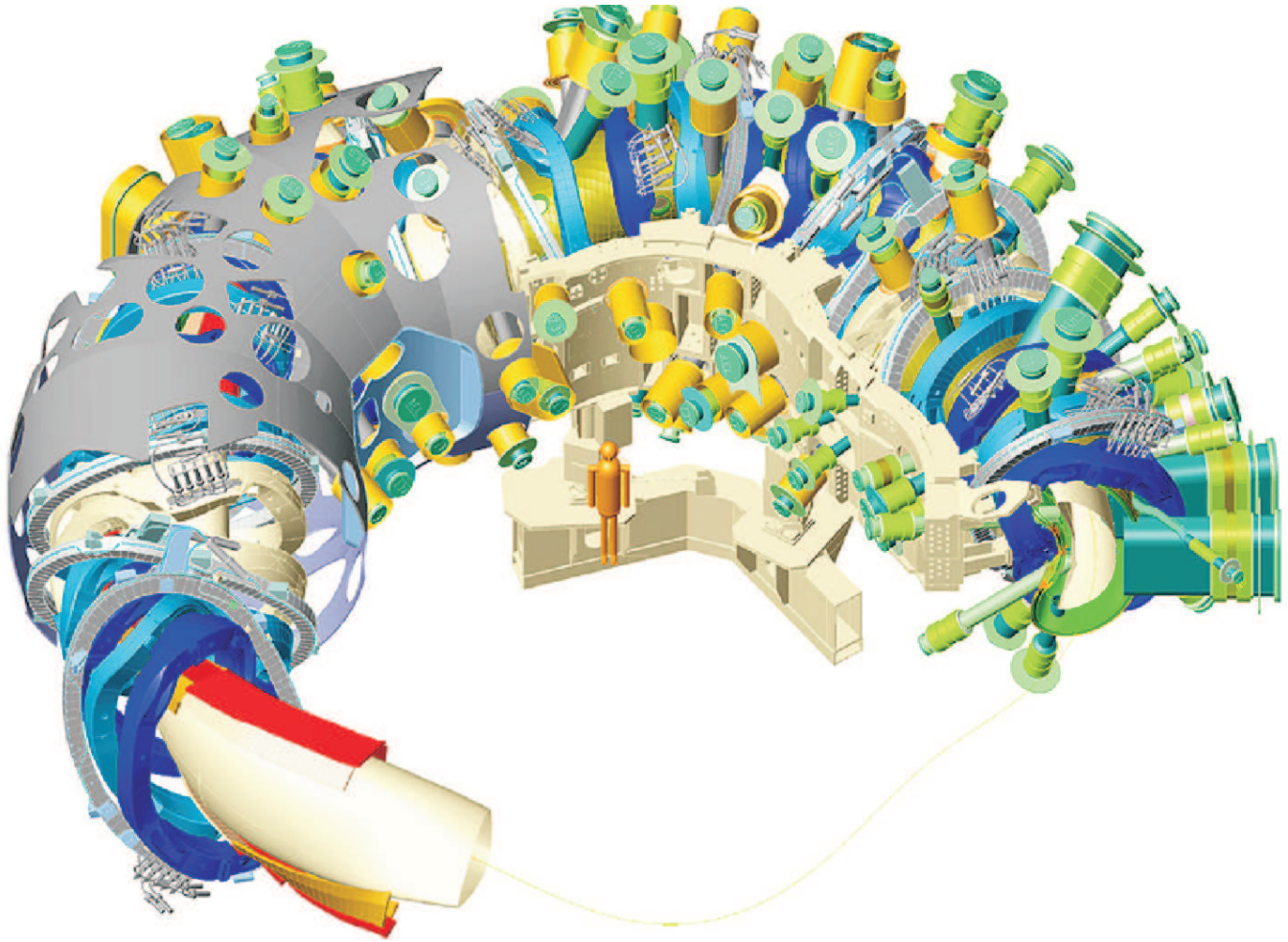
1. Reduce the problem
2. Devise algorithm
3. Create code
4. Apply code

Each step may raise new questions, unanticipated problems resulting in a return to earlier stages of the process.

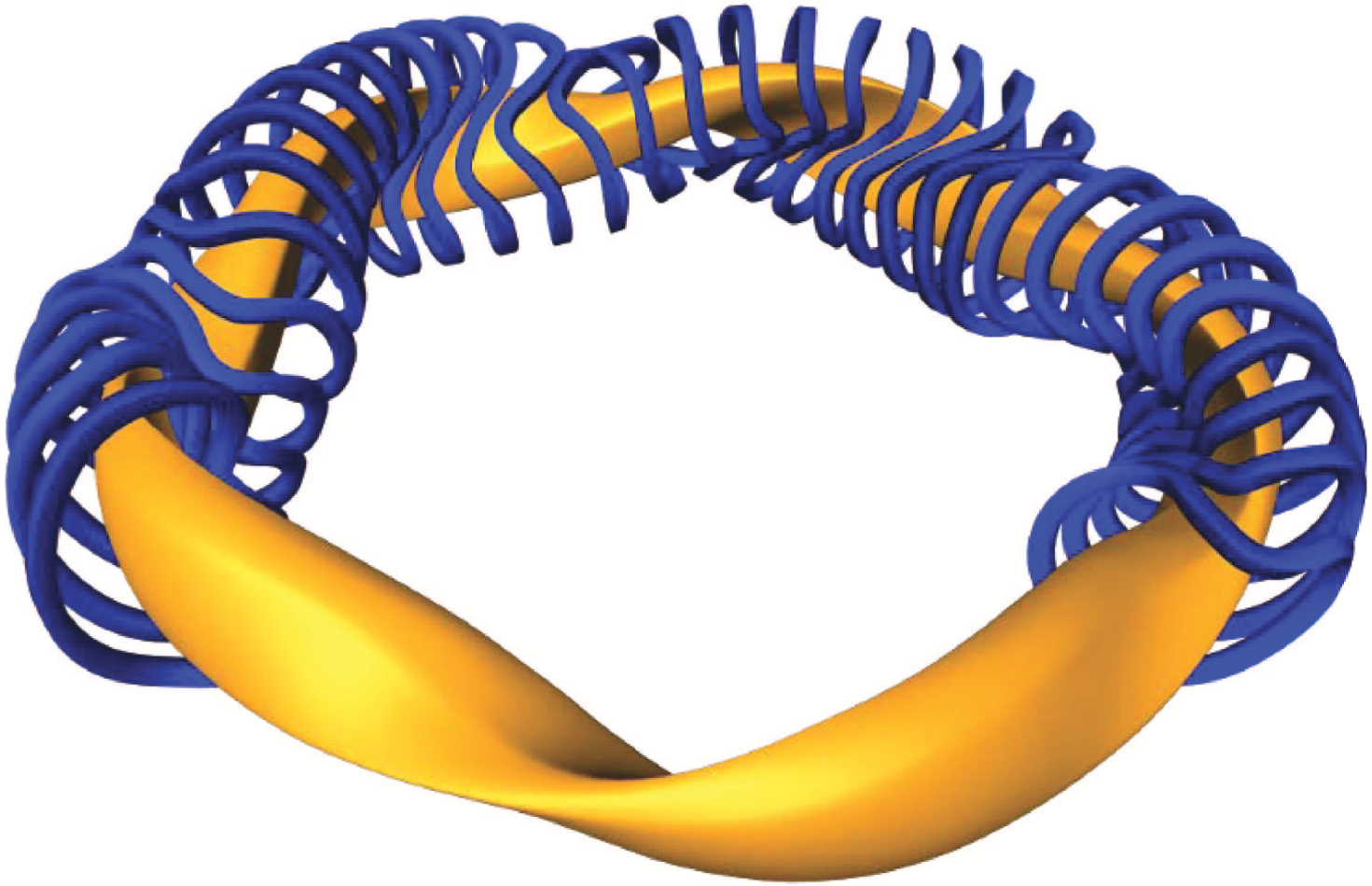
This phenomenon is representative for virtually all of science.



Computational Physics Applied: W7-X



Computational Physics Applied: W7X



Numerical Mathematics: Algorithms matter

Algorithms decisively affect accuracy, performance and scope of a code.

- Library functions are often preferred over homebrew algorithms - but not always!
- Try to exploit special properties of your problem

The gains made with ever improving algorithms are comparable to those obtained from the progress of IT technology (Moores law).

Inefficient algorithms get left behind...

Algorithms, Example: Field Line Tracing

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(t,x), \qquad f(t,x) = \frac{\vec{B}}{\left|\vec{B}\right|}$$

Naive approach: explicit single step (Euler-Cauchy)

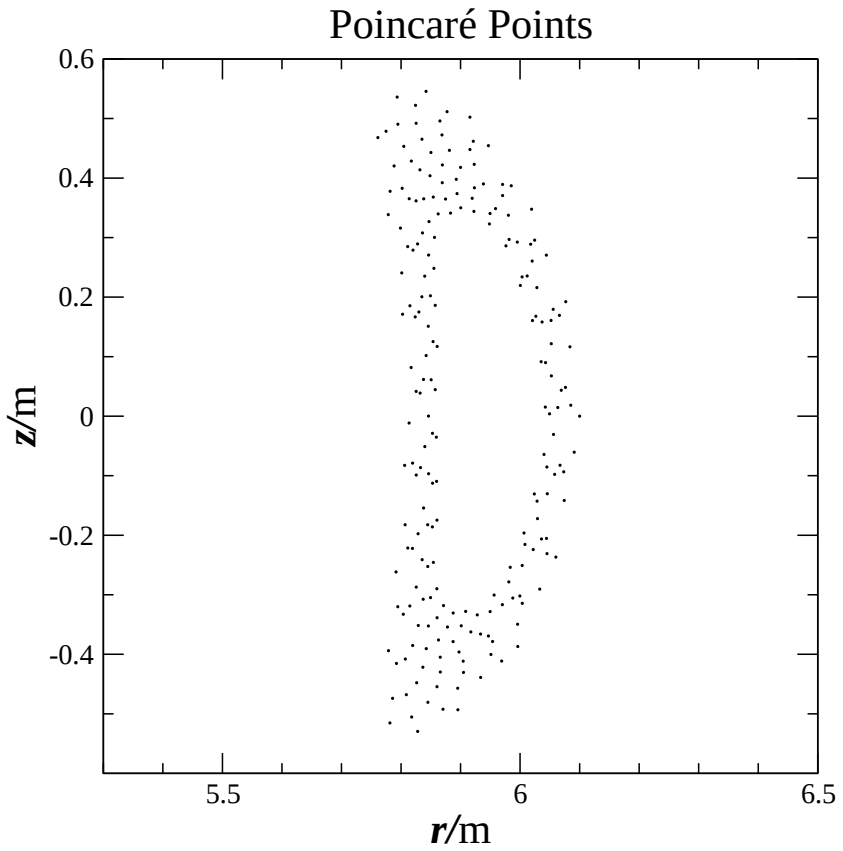
$$x_{n+1} = x_n + f(t_n, x_n) \cdot \Delta t$$

Not too sophisticated, but much better: Adams-Bashforth

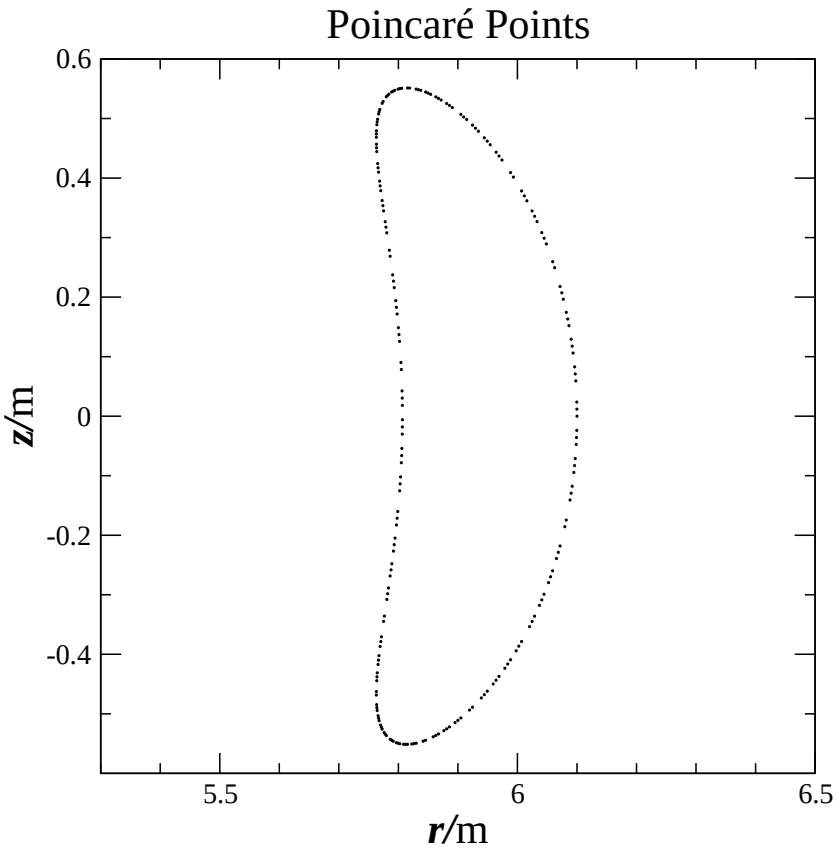
$$x_{n+1} = x_n + \left(\frac{55}{24} f(x_n) + \frac{-59}{24} f(x_{n-1}) + \frac{37}{24} f(x_{n-2}) + \frac{-9}{24} f(x_{n-3}) \right) \cdot \Delta t$$

Algorithms Matter: Field Line Tracing

Euler-Cauchy



Adams-Bashforth



step size = 0.5mm

Algorithms Matter: Monte Carlo Integration

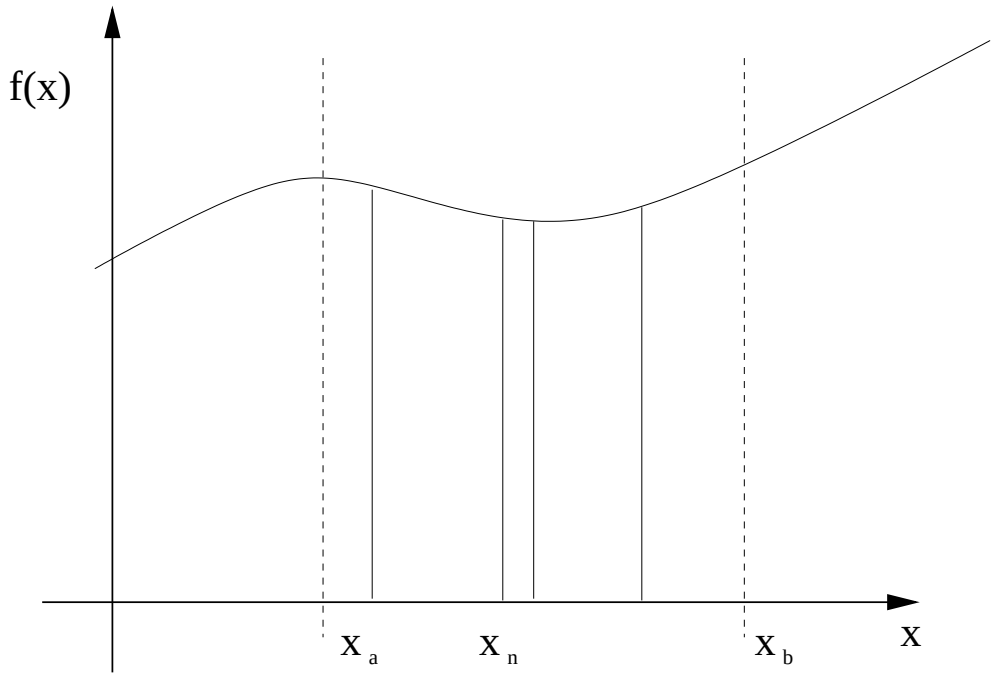
Problem: integrate

$$F = \int_{x_a}^{x_b} f(x)\mathrm{d}x$$

Monte-Carlo approach, naive:

$$F_N \approx \frac{x_b - x_a}{N} \sum_N f(x_n)$$

$$|F_N - F| \sim \frac{x_b - x_a}{\sqrt{N}}$$



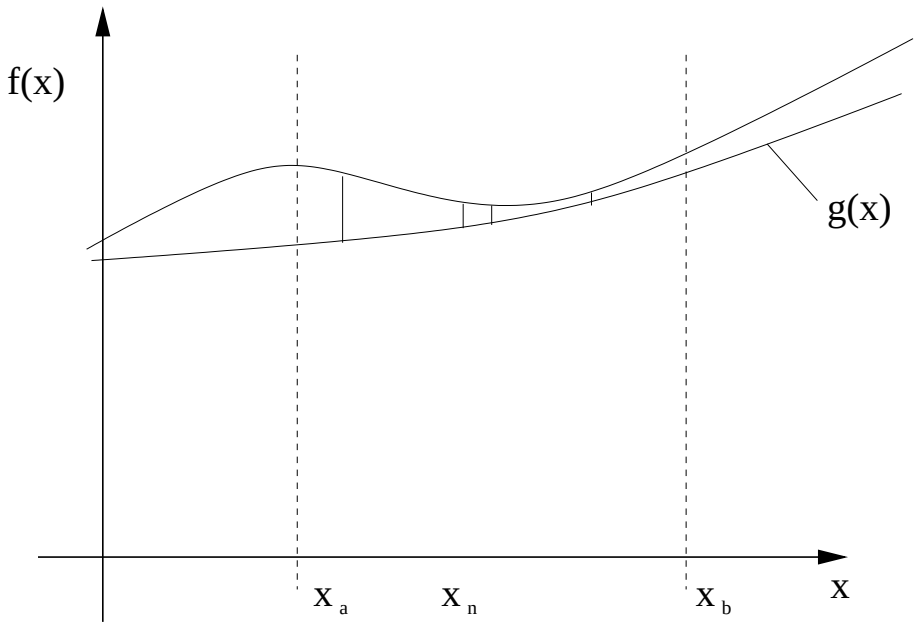
Algorithms Matter: Monte Carlo Integration

Modified Monte-Carlo approach,
"control variates":

$$G = \int_{x_a}^{x_b} g(x)\mathrm{d}x.$$

$$F_N \approx \frac{x_b - x_a}{N} \sum_N (f(x_n) - g(x_n)) + G$$

$$|(F_N - G_N) - (F - G)| \sim \frac{1}{\sqrt{N}}$$



Languages - Little Choice

- FORTRAN
  Appeared 1957, refuses to die
  High degree of standardisation
  Language standard revisions FORTRAN ... Fortran V ... FORTRAN2003
  Vast amounts of numerical libraries written in FORTRAN

- C
  Developed 1972
  High degree of standardisation
  Covers all aspects of both structured and machine oriented programming
  Very common language

- C++
  Developed in 1980ies
  High degree of standardisation
  Combines features of C with object oriented concepts

Why FORTRAN/C/C++

- FORTRAN/C/C++ are common languages.
  Mature and efficient compilers are available on all relevant platforms.
- Used judiciously, all of these languages deliver good performance.
- All of these languages are able to use libraries
  written in FORTRAN or C.
- Standardisation guarantees portability to other platforms.

  Always comply with the language standard!

Numerical Libraries - Why?

A library is a collection of software (functions, classes) designed to assist the solution of a specific range of problems.

- library functions can save a lot of work
- they have been tested by a large community of users and are less likely to suffer from programming glitches.
- many library functions are the result of man-months, maybe man-years of cutting edge mathematical research. Most of them are hard to beat, both in performance and accuracy.
- The BLAS (Basic Linear Algebra Subroutines) library is available in taylor made versions optimised for individual computer architectures and even processors.

Libraries: Important Examples

Name of Library	Purpose	Description	Availability
BLAS	Low level linear algebra	Basic vector/ matrix operations	free, often optimised for indiv. platforms, `www.netlib.org/`
LAPACK	Higher level linear algebra	Eigenvalue problems, linear equations, ...	free `www.netlib.org`
GSL (GNU Scientific Library)	General purpose	extensive, growing	free (`www.gnu.org`) C, ∃ wrappers for other languages
PETSc	PDEs, ODEs, linear algebra, non-lin. equations, ...	Parallel	free `www.mcs.anl.gov`
NETLIB	general purpose	very extensive various languages	free, www.netlib.org
IMSL (International Mathematical and Statistical Library)	General purpose	very extensive, dates back to 1970ies	commercial, C, Java, C#, FORTRAN
NAG (Numerical Algorithms Group)	General purpose	very extensive dates back to 1970ies	commercial, FORTRAN, C/C++

EXAMPLE: NAG Library, Chapters

A00 Library Identification
A02 Complex Arithmetic
C02 Zeros of Polynomials
C05 Roots of One or More Transcendental Equations
C06 Summation of Series
D01 Quadrature
D02 Ordinary Differential Equations
D03 Partial Differential Equations
D04 Numerical Differentiation
D05 Integral Equations
D06 Mesh Generation
E01 Interpolation
E02 Curve and Surface Fitting
E04 Minimizing or Maximizing a Function
F Linear Algebra

EXAMPLE: NAG Library, Chapters

F01 Matrix Operations, Including Inversion
F02 Eigenvalues and Eigenvectors
F03 Determinants
F04 Simultaneous Linear Equations
F05 Orthogonalisation
F06 Linear Algebra Support Routines
F07 Linear Equations (LAPACK)
F08 Least-squares and Eigenvalue Problems (LAPACK)
F11 Large Scale Linear Systems
F12 Large Scale Eigenproblems
G01 Simple Calculations on Statistical Data
G02 Correlation and Regression Analysis
G03 Multivariate Methods
G04 Analysis of Variance
G05 Random Number Generators

EXAMPLE: NAG Library, Chapters

G07 Univariate Estimation
G08 Nonparametric Statistics
G10 Smoothing in Statistics
G11 Contingency Table Analysis
G12 Survival Analysis
G13 Time Series Analysis
H Operations Research
M01 Sorting
P01 Error Trapping
S Approximations of Special Functions
X01 Mathematical Constants
X02 Machine Constants
X03 Inner Products
X04 Input/Output Utilities
X05 Date and Time Utilities

Example: NAG Library - Quadrature (1)

Routine Name	Purpose
D01AHF	One-dimensional quadrature, adaptive, finite interval, strategy due to Patterson, suitable for well-behaved integrands
D01AJF	One-dimensional quadrature, adaptive, finite interval, due to Piessens & de Doncker, allowing for badly behaved integrands
D01AKF	One-dimensional quadrature, adaptive, finite interval, suitable for oscillating functions
D01ALF	One-dimensional quadrature, adaptive, finite interval, allowing for singularities at user-specified break-points
D01AMF	One-dimensional quadrature, adaptive, infinite or semi-infinite interval
D01ANF	One-dimensional quadrature, adaptive, finite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
D01APF	One-dimensional quadrature, adaptive, finite interval, weight function with end-point singularities of algebraico-logarithmic type
D01AQF	One-dimensional quadrature, adaptive, finite interval, weight function 1/(x-c), Cauchy principal value (Hilbert transform)
D01ARF	One-dimensional quadrature, non-adaptive, finite interval with provision for indefinite integrals

Example: NAG Library - Quadrature (2)

Routine Name	Purpose
D01ASF	One-dimensional quadrature, adaptive, semi-infinite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
D01ATF	One-dimensional quadrature, adaptive, finite interval, variant of D01AJF efficient on vector machines
D01AUF	One-dimensional quadrature, adaptive, finite interval, variant of D01AKF efficient on vector machines
D01BAF	One-dimensional Gaussian quadrature
D01BBF	Pre-computed weights and abscissae for Gaussian quadrature rules, restricted choice of rule
D01BCF	Calculation of weights and abscissae for Gaussian quadrature rules, general choice of rule
D01BDF	One-dimensional quadrature, non-adaptive, finite interval
D01DAF	Two-dimensional quadrature, finite region
D01EAF	Multi-dimensional adaptive quadrature over hyper-rectangle, multiple integrands
D01FBF	Multi-dimensional Gaussian quadrature over hyper-rectangle
D01FCF	Multi-dimensional adaptive quadrature over hyper-rectangle
D01FDF	Multi-dimensional quadrature, Sag-Szekeres method, general product region or n-sphere

Example: NAG Library - Quadrature (3)

Routine Name	Purpose
D01GBF	Multi-dimensional quadrature over hyper-rectangle, Monte Carlo method
D01GCF	Multi-dimensional quadrature, general product region, number-theoretic method
D01GDF	Multi-dimensional quadrature, general product region, number-theoretic method, variant of D01GCF efficient on vector machines
D01GYF	Korobov optimal coefficients for use in D01GCF or D01GDF, when number of points is prime
D01GZF	Korobov optimal coefficients for use in D01GCF or D01GDF, when number of points is product of two primes
D01JAF	Multi-dimensional quadrature over an n-sphere, allowing for badly behaved integrands
D01PAF	Multi-dimensional quadrature over an n-simplex

Libraries Summary

- Libraries written in FORTRAN or C can easily be called from FORTRAN, C or C++.
- FORTRAN2003 introduces communication with C++.
- Don't mix too many different libraries.
- Use those libraries you expect to be available on platforms you might have to port to.
- Libraries cover a wide range of standard applications
- Most libraries are well tested and often employ cutting edge methods

The computational scientist will have plentyful opportunity to write his own numerical algorithms, anyway.

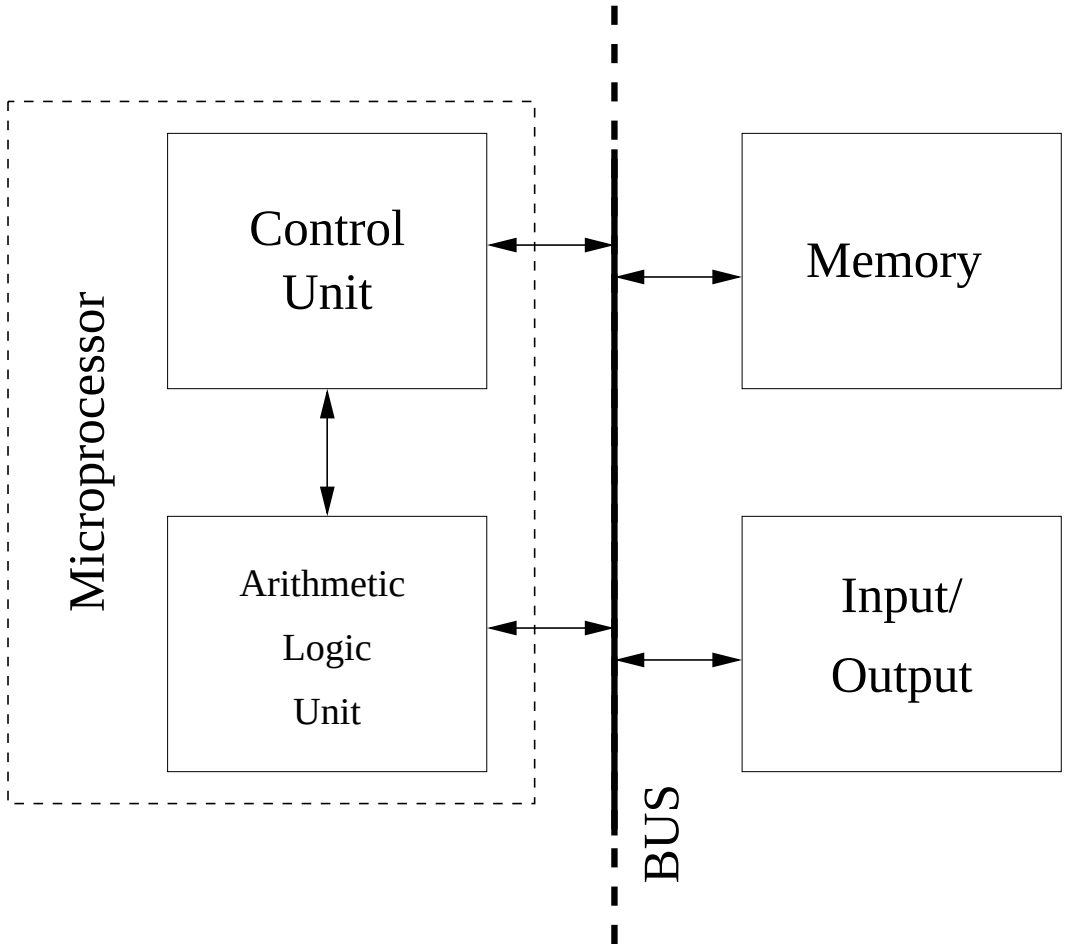
Computers ..... Come in Different Flavours!

- Scalar architecture (bottom end PC $\Rightarrow$ irrelevant for HPC!)
- Multicore architecture (regular PC, workstation)
- Parallel architecture
  - Cluster: multiple computer
  - Blade server: multiple board in rack
  - regular GPU
- Vector architecture

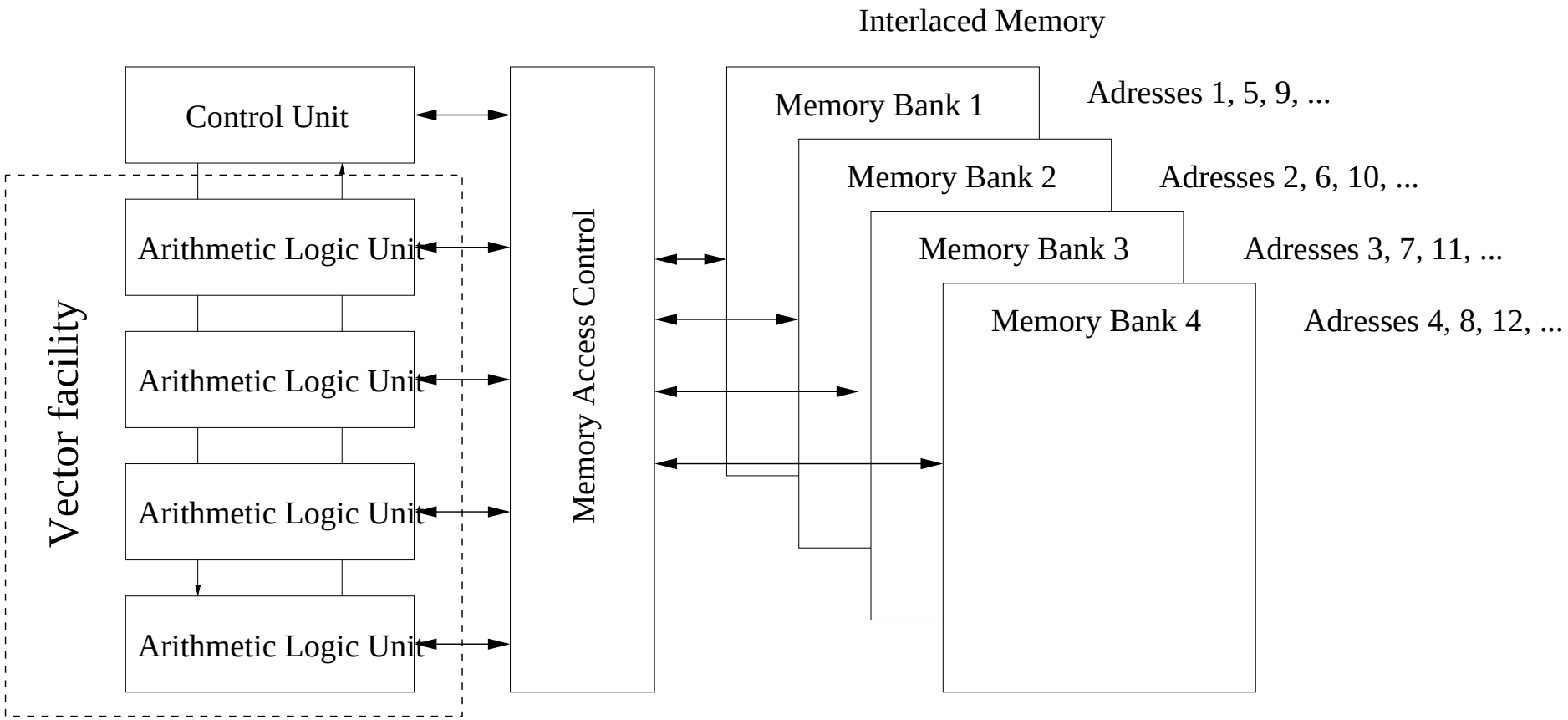
Different computer architectures require different programming styles/tools in order to get performance/scaleability.

Scalar Computers

- Von Neumann architecture
- data and instructions in same memory space
- common and cheap device

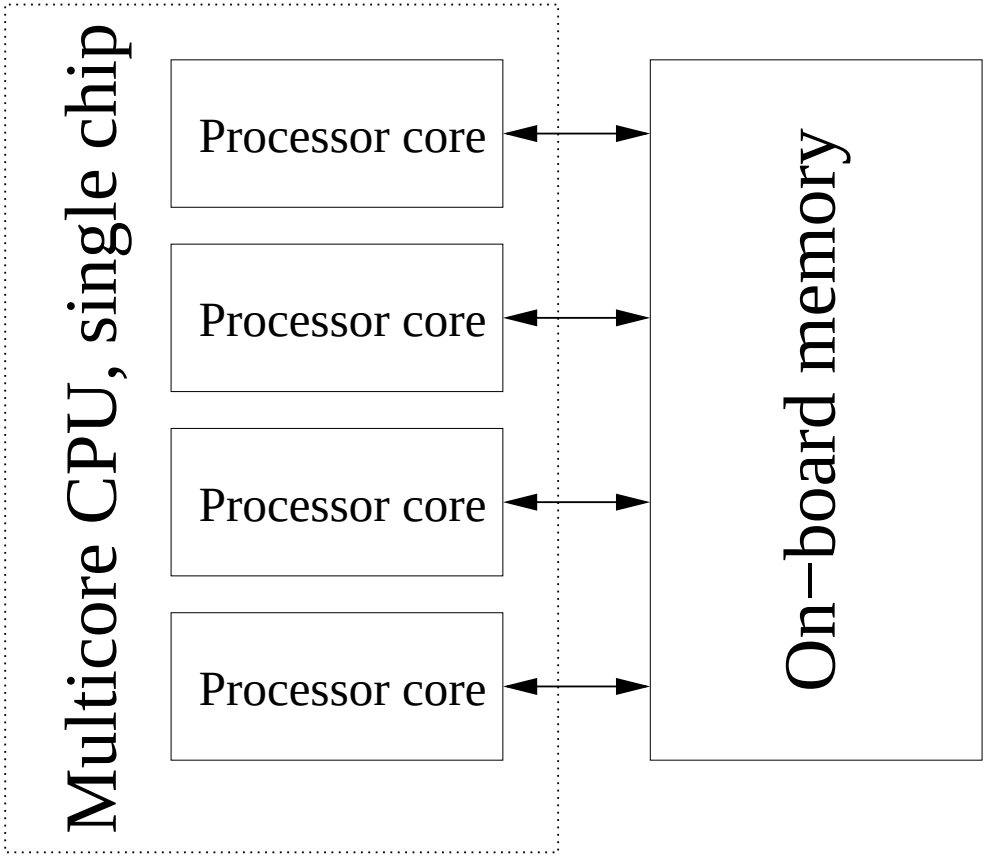


Vector Computers

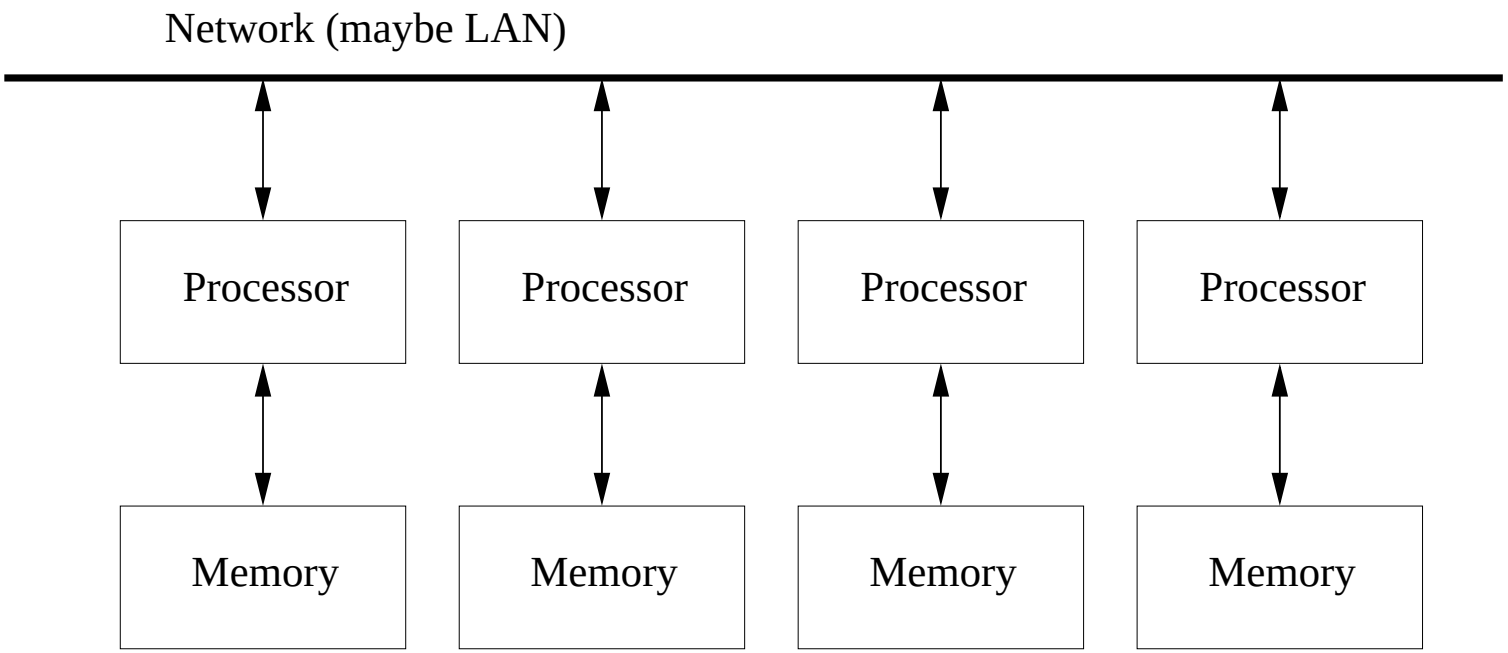


Parallel Computers: Multicore PC

- several Von Neumann type processor cores on same chip
- cores share same memory
- cores communicate via shared memory

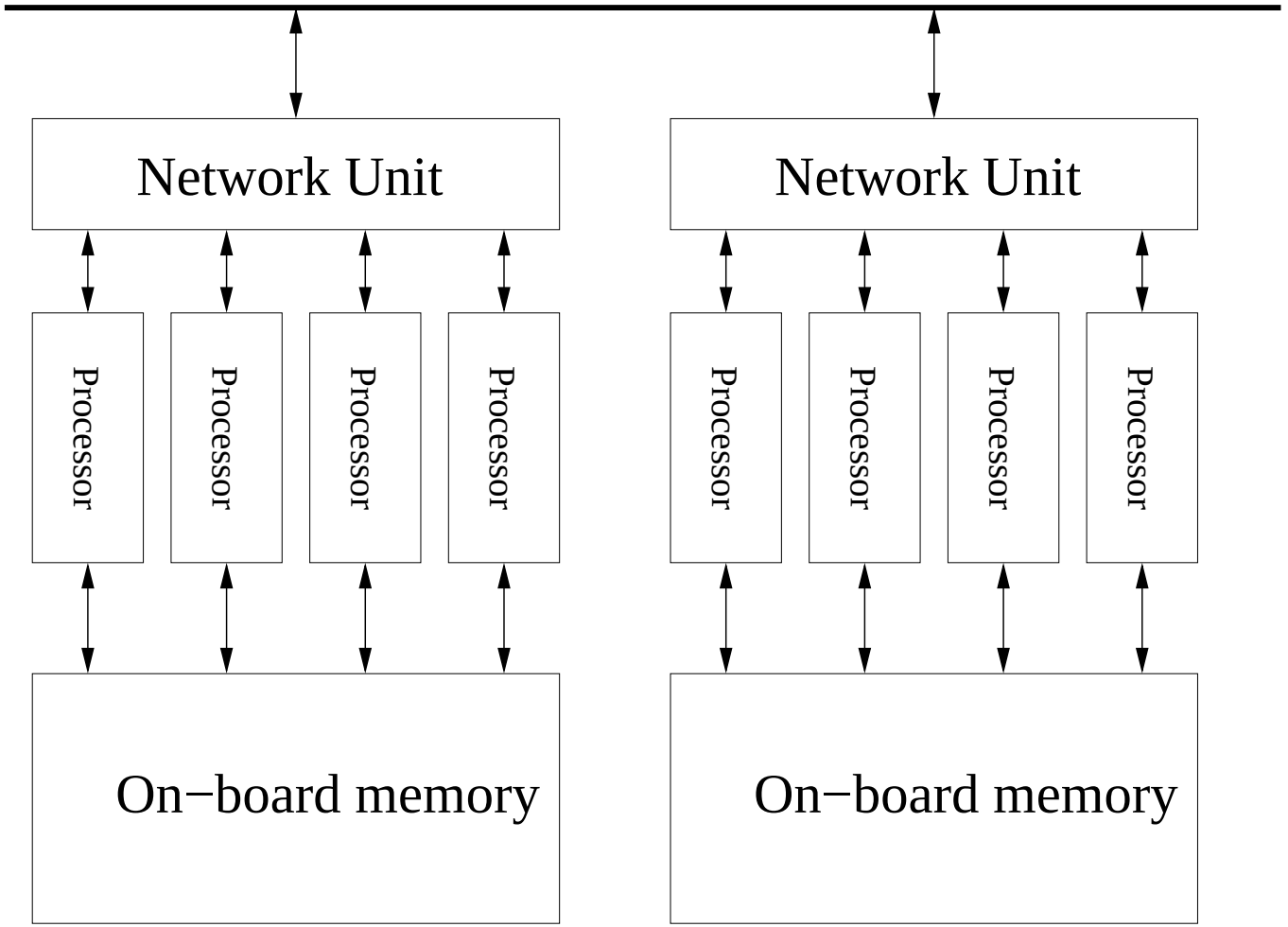


Parallel Computers: Cluster

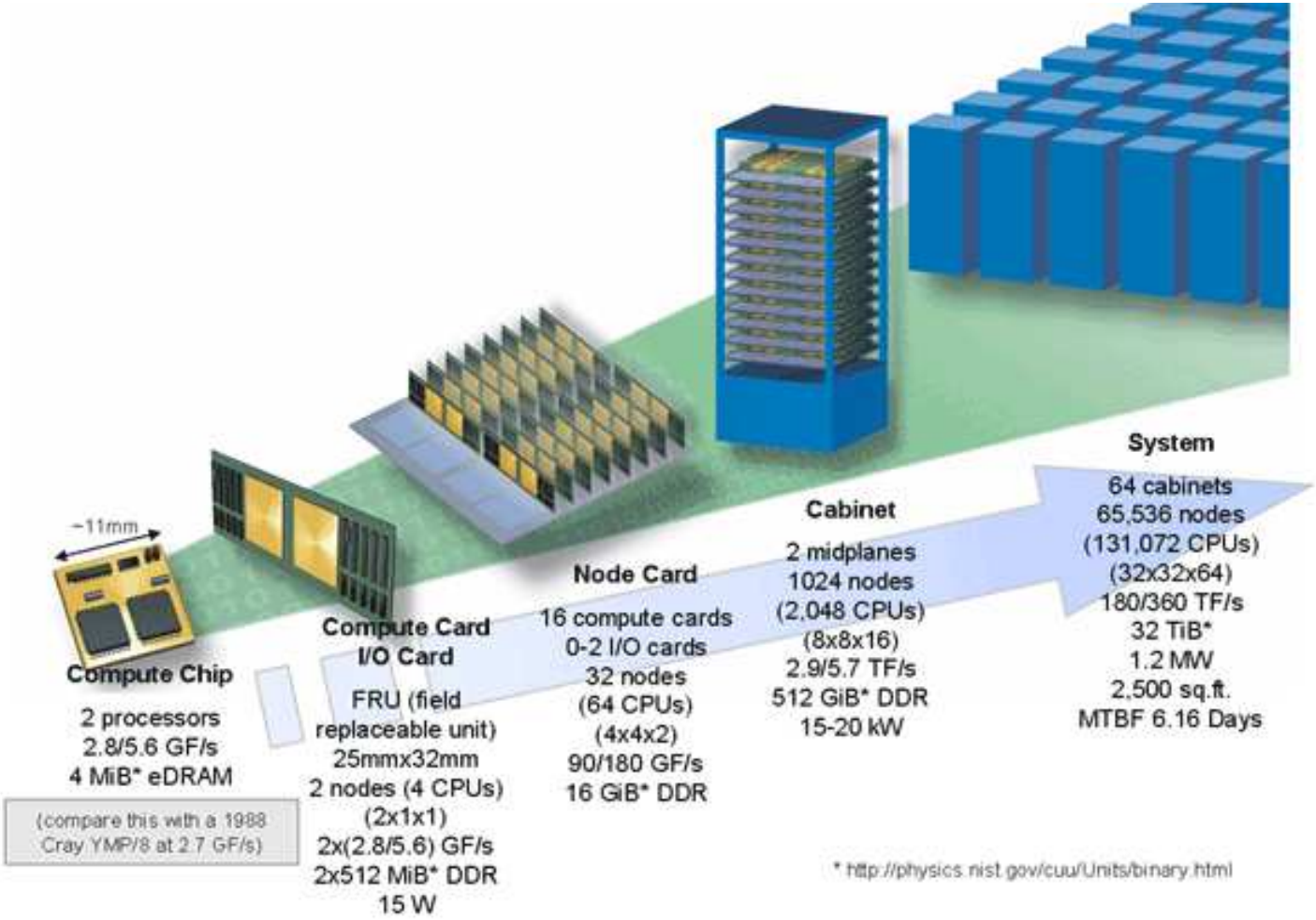


- abundant computing power at moderate investment cost
- Network may turn out communication bottleneck, resulting in poor scaleability

Parallel Computers: Blade Server

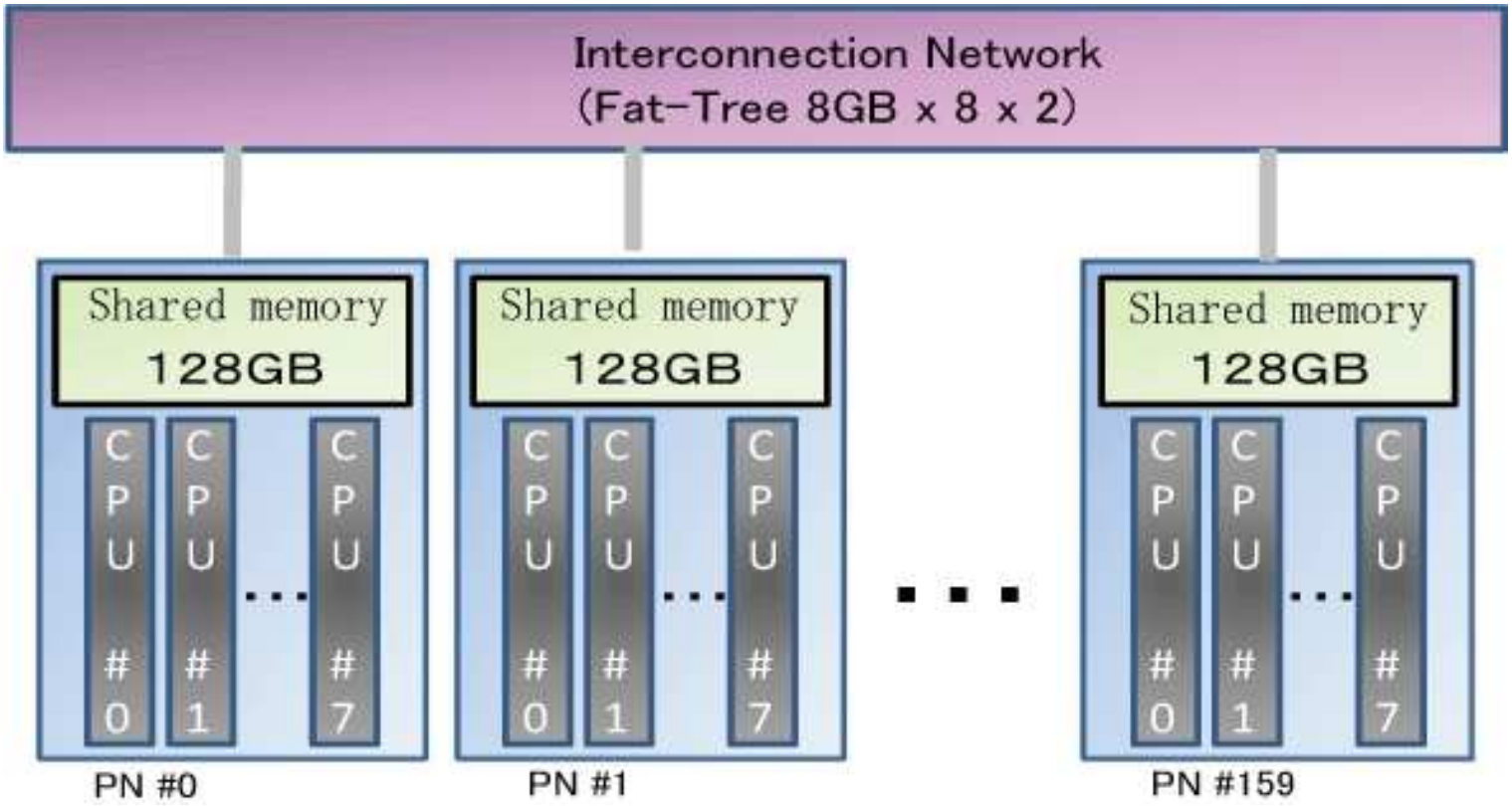


Parallel Computers Example: Blue Gene



© Wikipedia

Vector Computers Example: Earth Simulator 2



165×8 PEs, 128GB shared mem. $\Rightarrow$ 131TFlops

... and the vector computer, too, is parallel!

Vector Computers Example: Earth Simulator 2

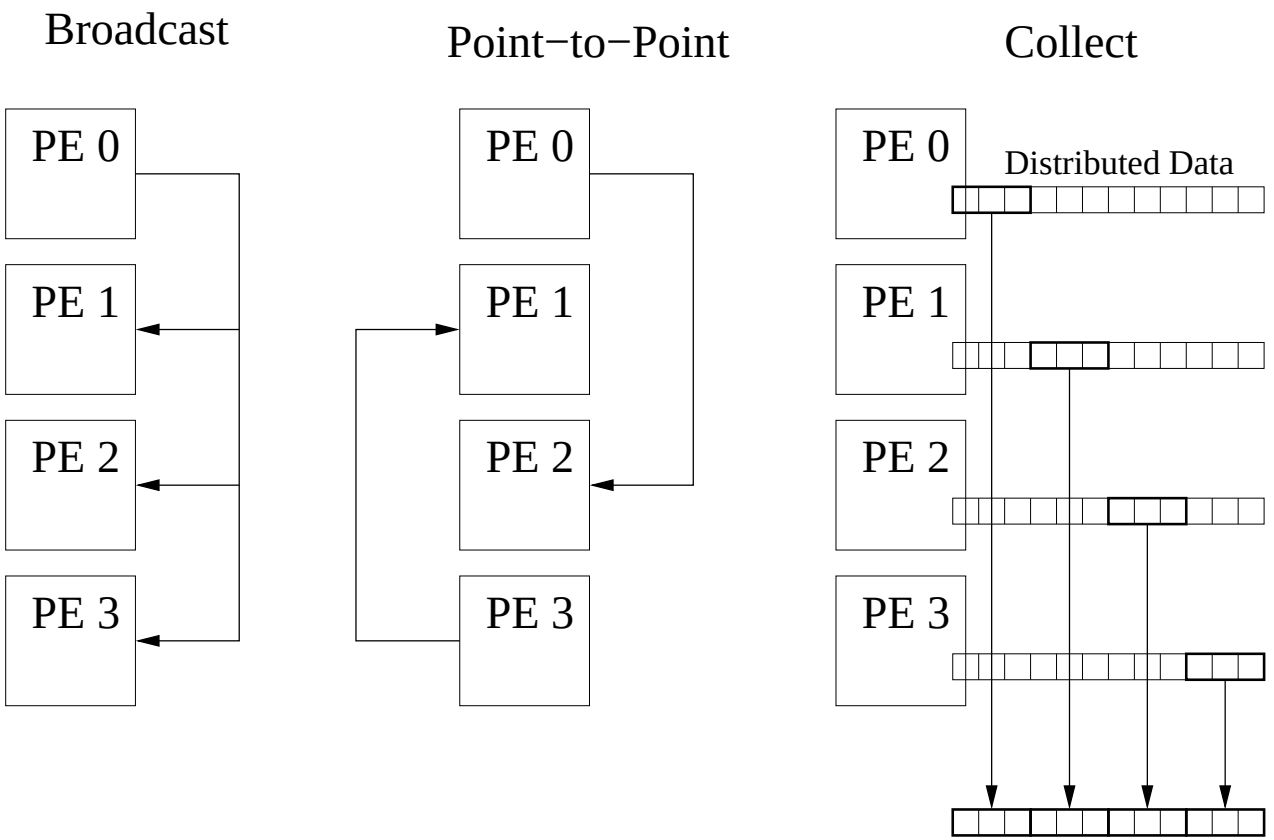
Coming out of fashion...

Computers at MPCDF - they're all parallel!

- IBM iDataPlex (Hydra)
  - Intel Ivy Bridge based cluster with 3500 nodes,
    20 cores @ 2.8 GHz / node, 64-128 GB / node, 200 TFlop/s total
  - Intel Sandy Bridge-EP based cluster with 610 nodes,
    20 cores @2 2.8 GHz / node, 64GB / node, 200 TFlop/s total
  - 338 nodes with 2 NVIDIA K20X GPGPUs each,
    12 nodes with 2 Intel Xeon Phi cards each
- Hydra Extensiuon Cluster
  - Intel Xeon E5-2698 processors with 880 nodes, 32 cores @ 2.3 GHz / node, 128 - 512GB / node
  - 106 nodes equipped w. GPU cards (2 x PNY GTX980 each).
  - 4 login nodes + 8 I/O nodes serving 1.5 PB HDD.
- Dedicated Linux clusters
  24000 CPU cores (total) operated for 14 Max Planck Institutes / groups
- Visualisation
  4 Hydra GPU nodes configured for remote visualization, 8 visualization slots
  2 Intel Xeon CPUs (E5-2680 @ 2.80GHz), 10 cores each, 128 - 256 GB
  2 NVIDIA Tesla GPUs (K20x), 6 GB GPU each

Message Passing Interface (MPI)

MPI is a library providing functions for network based communication between processors. Here are some typical operations...



MPI: Example


```
#include "mpi.h"
int main(int argc, char **argv)
   {char message[20];
    int myrank;
    MPI_Status status;

    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &myrank);

    if (myrank == 0)
       {strcpy (message, "Hello, there");
        MPI_Send(message, strlen(message), MPI_CHAR, 1, 99, MPI_COMM_WORLD);
       }
    else
       {MPI_Recv(message, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status);
        printf("received :%s:\n", message);
       }
    MPI_Finalize();
   }
```

OpenMP: Example


```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
   {omp_set_num_threads(4);
#pragma omp parallel for
    for (int i = 0; i < 4; ++i)
       {int id = omp_get_thread_num();
        printf("Hello World from thread %d\n", id);
#pragma omp barrier
        if (id == 0)
            printf("There are %d threads\n", omp_get_num_threads());
       }
    return EXIT_SUCCESS;
   }
```

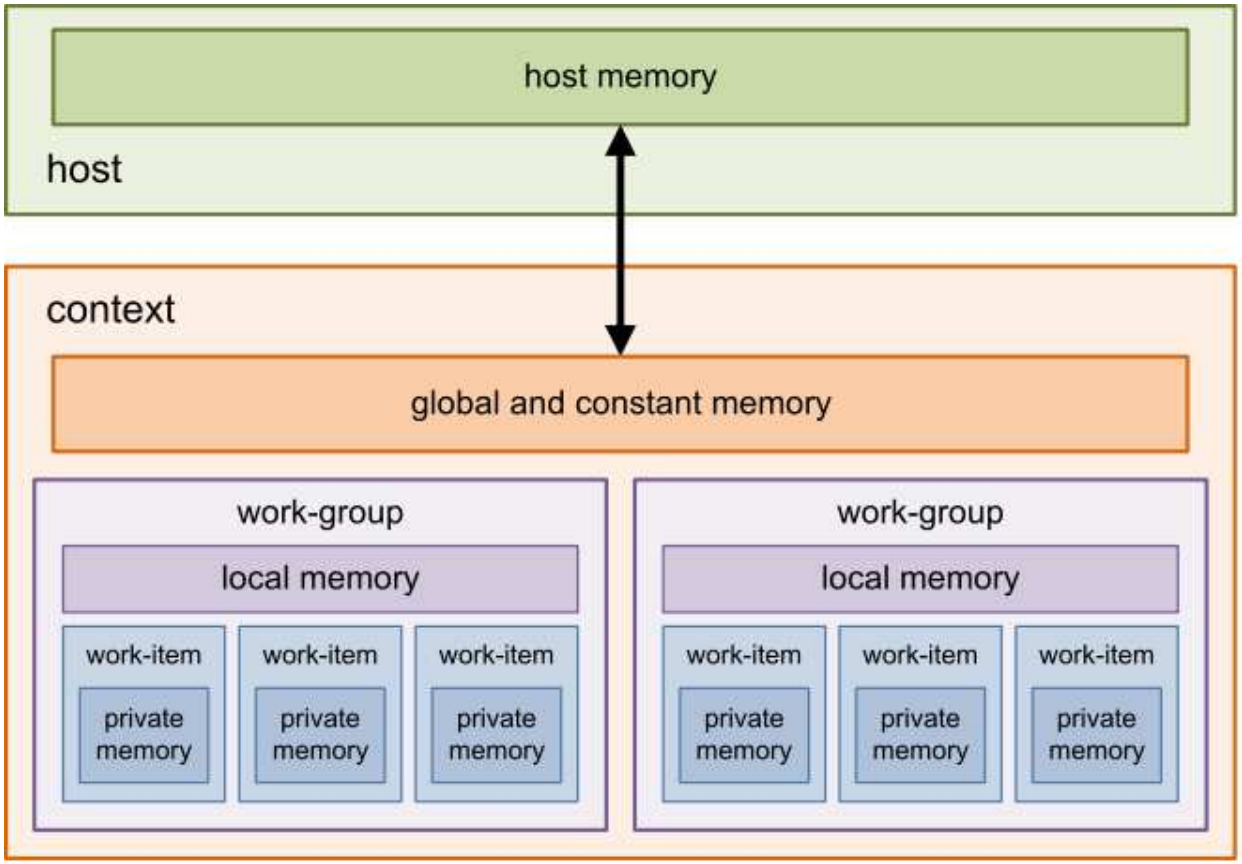
MPI vs. OpenMP

- MPI suited for (network based) communication between computer boards. Typical applications: master/slave or division of loops.

- OpenMP suited for (on board) communication via shared memory Typical application: divide loops between PE cores.
- large applications on big systems may require both.

Parallel computing: GPU programming (Background)

Power consumption of PE increases with 3rd power of f_{clock}.
Computational requirements in CG solved by parallelisation $\Rightarrow 10^3$ PEs



GPU programming models

- ∃ OpenCL (Khronos) and CUDA (NVIDIA)
- two codes:
  - main code running on Host: C/C++ and API ⇒ device
  - kernel code running on device very similar to C, with restrictions
- 1 Kernel running on 1 PE (Work Item) at a time
- ∃ communication between kernels/PEs in compute unit/work group

Restrictions for Kernel C

- no recursion
- cannot pass pointer to pointers
- return type must be `void`
- no `extern`, `static`, `auto`, `register`
- many standard headers & lib. functions not supported:
  `stdio.h`, `string.h`, `stdlib.h`, `stdarg.h`, ...
- no dynamic memory allocation within kernel code
- ...

OpenCL Work Flow

1. Identify suitable device(s) in Platform
2. Host loads and compiles kernel code (during runtime)
3. Host creates input data, tells device to generate memory objects (allocation)
4. Host dispatches kernel code to device queue

Code Example OpenCL: Vector Addition in C

```
void vector_add (long n,
                 const float *a,
                 const float *b,
                 float *sum)
   {long i;
    for (i=0; i<n; i++) sum[i] = a[i] + b[i];
   }
```

simple in C

Code Example OpenCL: Vector Addition Kernel Code

```
__kernel void vectorAdd( __global const float * a,
                         __global const float * b,
                         __global float * c)
   {
// Vector element index
    int nIndex = get_global_id(0);
    c[nIndex] = a[nIndex] + b[nIndex];
   }
```

Example OpenCL: (simplified) Vector Add Host Code (1)

```
const unsigned int cnBlockSize = 512;
const unsigned int cnBlocks = 3;
const unsigned int cnDimension = cnBlocks * cnBlockSize;

char *sProgramSource;    //.... add code to read source to string

// create OpenCL device & context
cl_context hContext;
hContext = clCreateContextFromType(0, CL_DEVICE_TYPE_GPU, 0, 0, 0);

// query all devices available to the context
size_t nContextDescriptorSize;
clGetContextInfo(hContext, CL_CONTEXT_DEVICES,0, 0,
                 &nContextDescriptorSize);
cl_device_id * aDevices = malloc(nContextDescriptorSize);
clGetContextInfo(hContext, CL_CONTEXT_DEVICES,
                 nContextDescriptorSize, aDevices, 0);
```

Example OpenCL: (simplified) Vector Add Host Code (2)

```
// create a command queue for first device the context reported
cl_command_queue hCmdQueue;
hCmdQueue = clCreateCommandQueue(hContext, aDevices[0], 0, 0);

// create & compile program
cl_program hProgram;
hProgram = clCreateProgramWithSource(hContext, 1, sProgramSource, 0, 0);
clBuildProgram(hProgram, 0, 0, 0, 0, 0);

// create kernel
cl_kernel hKernel;
hKernel = clCreateKernel(hProgram, ''vectorAdd'', 0);
```

Example OpenCL: (simplified) Vector Add Host Code (3)


```
float * pA = new float[cnDimension];      // allocate host vectors
float * pB = new float[cnDimension];
float * pC = new float[cnDimension];

randomInit(pA, cnDimension);    // initialize host memory
randomInit(pB, cnDimension);

// allocate device memory
cl_mem hDeviceMemA, hDeviceMemB, hDeviceMemC;
hDeviceMemA = clCreateBuffer(hContext,
                             CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                             cnDimension * sizeof(cl_float), pA, 0);
hDeviceMemB = clCreateBuffer(hContext,
                             CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                             cnDimension * sizeof(cl_float), pB, 0);
hDeviceMemC = clCreateBuffer(hContext, CL_MEM_WRITE_ONLY,
                             cnDimension * sizeof(cl_float), 0, 0);
```

Example OpenCL: (simplified) Vector Add Host Code (4)

```
// setup parameter values
clSetKernelArg(hKernel, 0, sizeof(cl_mem), (void *)&hDeviceMemA);
clSetKernelArg(hKernel, 1, sizeof(cl_mem), (void *)&hDeviceMemB);
clSetKernelArg(hKernel, 2, sizeof(cl_mem), (void *)&hDeviceMemC);

// execute kernel
clEnqueueNDRangeKernel(hCmdQueue, hKernel, 1, 0, &cnDimension, ...);

// copy results from device back to host
clEnqueueReadBuffer(hContext, hDeviceMemC, CL_TRUE, 0,
                    cnDimension * sizeof(cl_float), pC, ...);

clReleaseMemObj(hDeviceMemA);   // release device memory
clReleaseMemObj(hDeviceMemB);
clReleaseMemObj(hDeviceMemC);
```

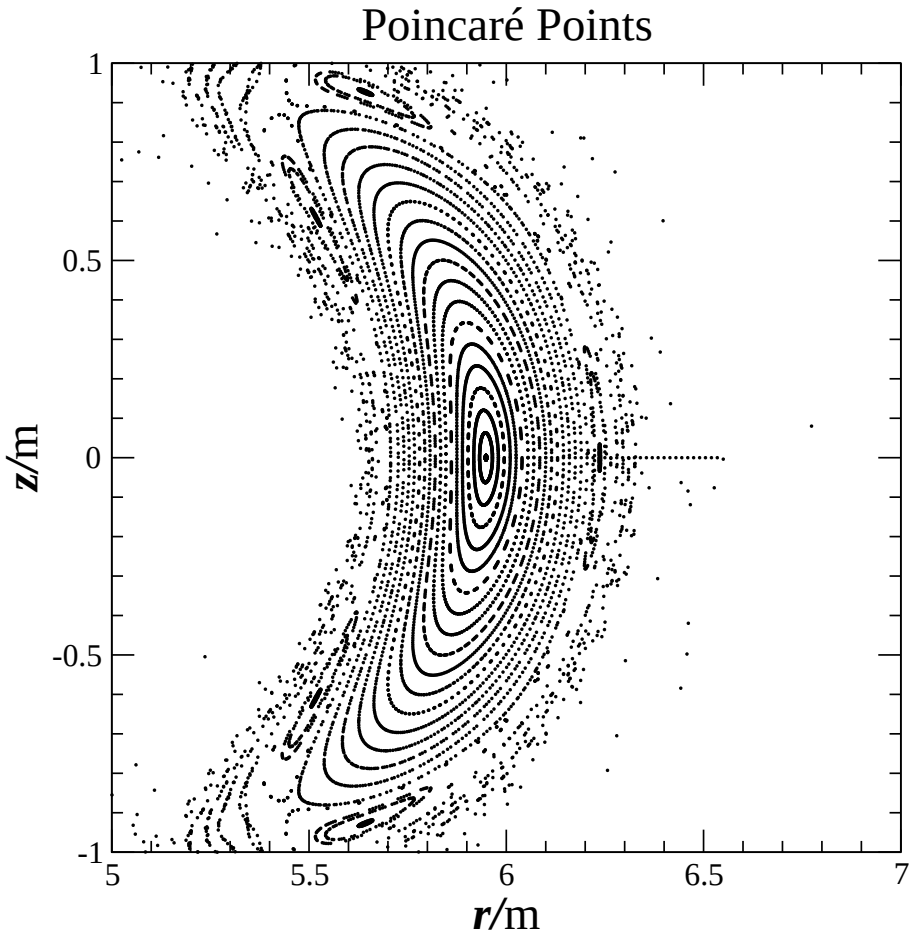
Code Examples, Tiny: GOURDON

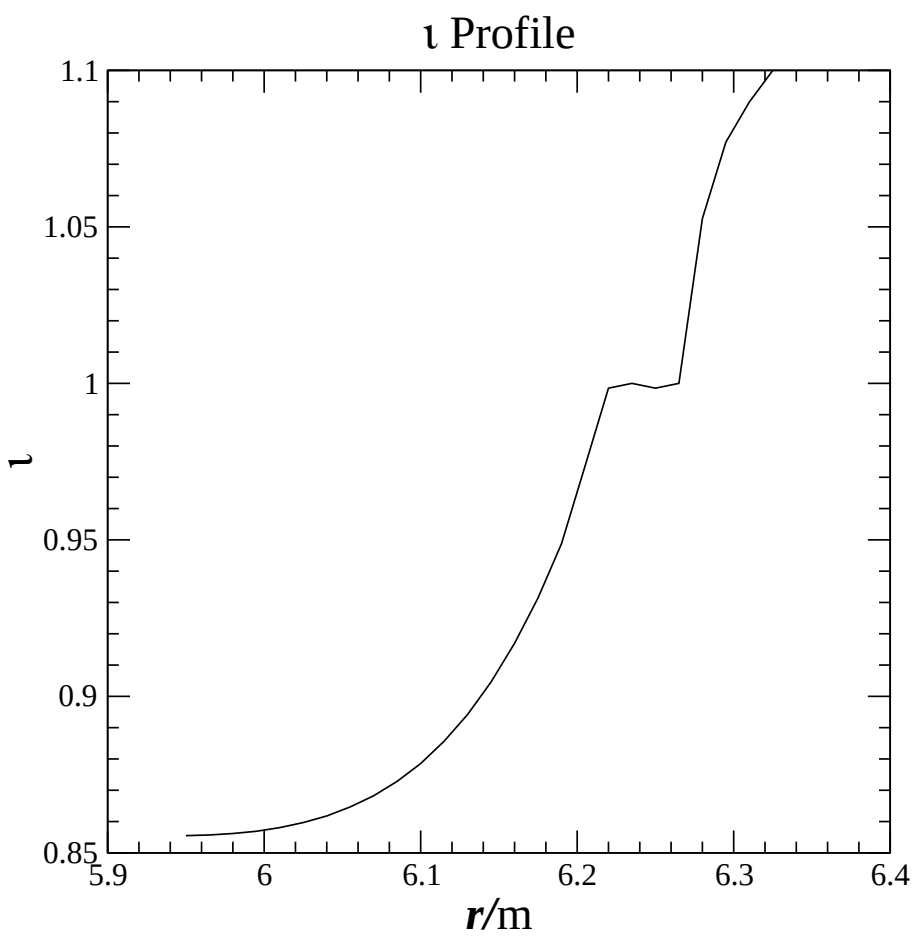
Written:	1968
First published:	1970
Authors:	C. Gourdon
Language:	FORTRAN66, now FORTRAN90
Libraries used:	/
Source:	$\sim$ 1200 lines
First platform:	at IPP: CRAY XMP
Execution time:	several minutes on 2016 PC

Code Examples, Tiny: GOURDON

- Computes magnetic field from current filaments using Biot-Savart rule.
- Integrates magnetic field lines.
- Generates Poincaré cuts.
- Computes field line parameters (rotational transform, mag. volume...).

Code Examples, Tiny: GOURDON





Code Examples, bigger: EUTERPE

Written:	derived from `GYGLES` written in 1996
First published:	2000
Authors:	G.Jost
Language:	FORTRAN90
Libraries used:	PETSc, FFTW
Source:	20000 lines
Purpose:	GK theory, ITG modes
First platform:	parallel UNIX/LINUX systems
Execution time:	typ 48h (or more) on 256 cores (HYDRA)

Code Examples, bigger: EUTERPE

The problem: describe ion motion in plasma core.
Vlasov equation:

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial t} + \vec{v} \cdot \nabla_r f + \frac{q}{M}(\vec{E} + \vec{v} \times \vec{B}) \cdot \nabla_v f = 0$$

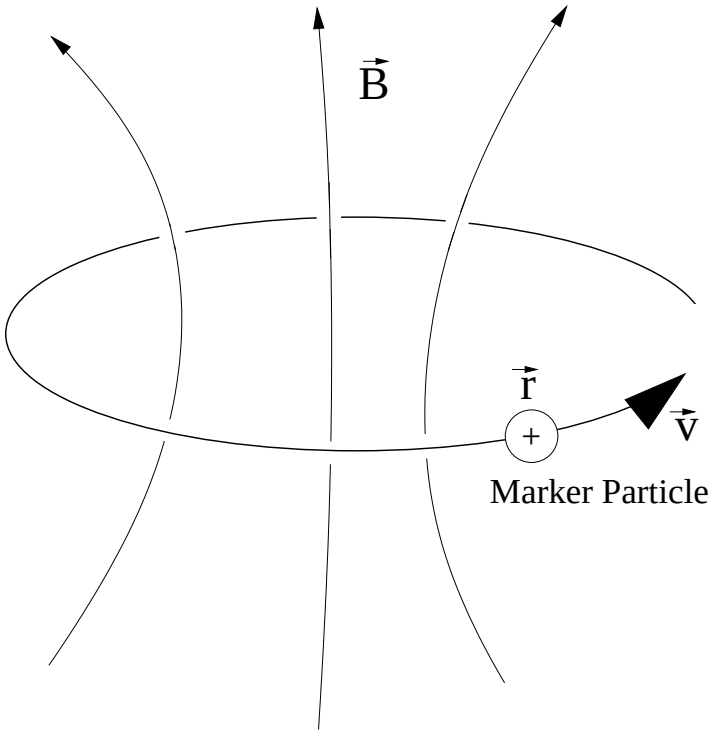
$f(\vec{r}, \vec{v}) =$ distribution function in 6d phase space.

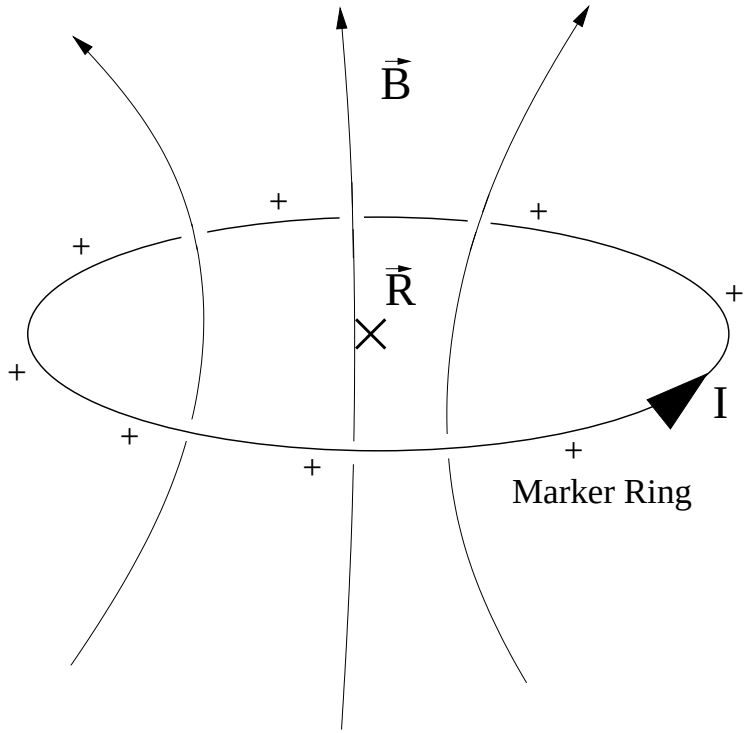
$\vec{E}(t) = -\nabla\phi(t)$ allowed to change.

$\vec{B}$ held constant, only external currents.

Code Examples, bigger: EUTERPE

Gyrokinetic approximation: eliminate gyroangle





EUTERPE: Gyrokinetic Equation

Simplification: average away gyromotion, but keep spatial dependence of fields.

$\Rightarrow$ describe particles as charged current loops with 5d distribution function

$$f(\vec{R}, v_{||}, \mu)$$

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial t} + \dot{\vec{R}} \cdot \nabla f + \dot{v}_{||} \frac{\partial f}{\partial v_{||}} + \dot{\mu} \frac{\partial f}{\partial \mu} = \mathcal{S}(f) = 0$$

$$\langle \phi \rangle (\vec{R}) := \frac{1}{2\pi} \int_0^{2\pi} \phi(\vec{R} + \vec{\rho}(\alpha))\, d\alpha$$

Code Examples, bigger: EUTERPE

- Gyrokinetic theory, **I**on **T**emperature **G**radient (ITG) modes
- Monte-Carlo approach
- δf method

1. Gyro-averaging: compute new forces on gyrorings$\sim$particles
2. Particle pushing: move MC particle markers
3. Charge assignment
4. Potential solver

EUTERPE: δf - Ansatz

$$f(t) = f_0 + \delta f(t)$$

$$\delta f = \sum_{p=1}^{N} \frac{1}{J} w_p(t)\, \delta(\vec{R} - \vec{R}_p(t))\, \delta(v_{\parallel} - v_{\parallel p}(t))\, \delta(\mu - \mu_p(t))$$

f_0 is assumed a Maxwellian

$$\frac{\mathrm{d}}{\mathrm{d}t}\delta f = -\mathcal{S}(f_0)$$

$$\dot{w}_p = -\mathcal{S}(f_0)\big|_{\vec{R}_p, v_{\parallel p}, \mu_p}$$

EUTERPE: Equations of Motion

$$\dot{R} = v_{\parallel}\vec{b} + \frac{1}{B}\vec{b}\times\nabla\langle\phi\rangle + \frac{\mu B + v_{\parallel}^2}{B\Omega_i}\vec{b}\times\nabla B$$

$$\dot{v}_{\parallel} = -\mu\,\vec{b}\cdot\nabla B - \frac{q}{M}\left(\vec{b} + \frac{v_{\parallel}}{B\Omega_i}\vec{b}\times\nabla B\right)\cdot\nabla\langle\phi\rangle$$

$$\dot{\mu} = 0$$

EUTERPE: Charge Assignment and Potential

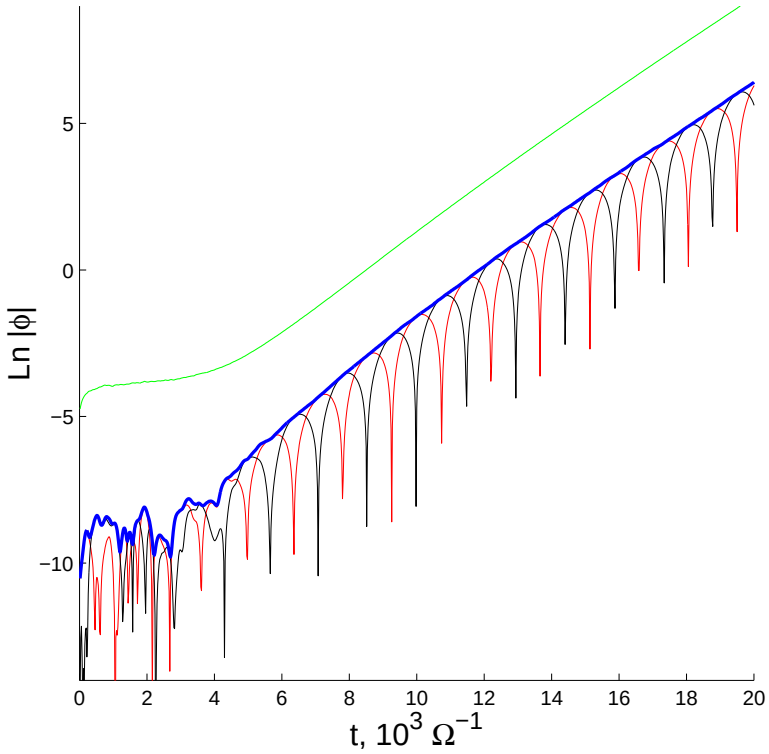
$$\phi(\vec{x}) = \sum_{\nu} \phi_\nu \Lambda_\nu(\vec{x})$$

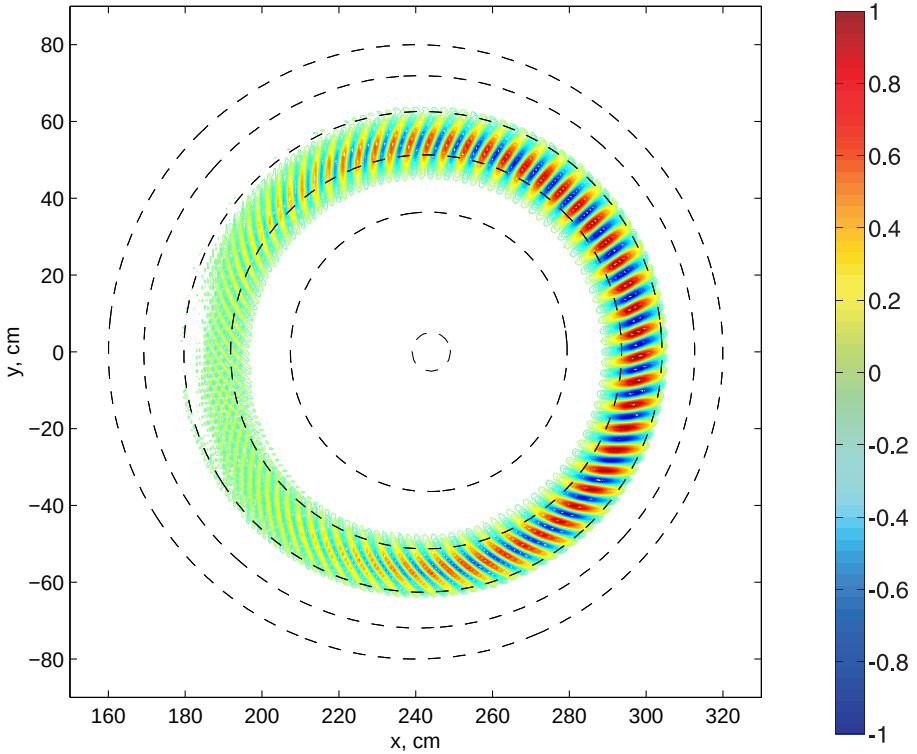
$$n_\nu = \sum_{p=1}^{N} w_p \frac{1}{2\pi} \int_0^{2\pi} \Lambda_\nu(\vec{R}_p + \vec{\rho}_p) d\alpha$$

$$A_{\nu\nu'} = \int \left(\frac{n_0}{B\Omega_i} \nabla_\perp \Lambda_\nu \cdot \nabla_\perp \Lambda_{\nu'} + \frac{e n_0}{k_\mathrm{B} T_\mathrm{e}} \Lambda_\nu \Lambda_{\nu'} \right) d\vec{x}$$

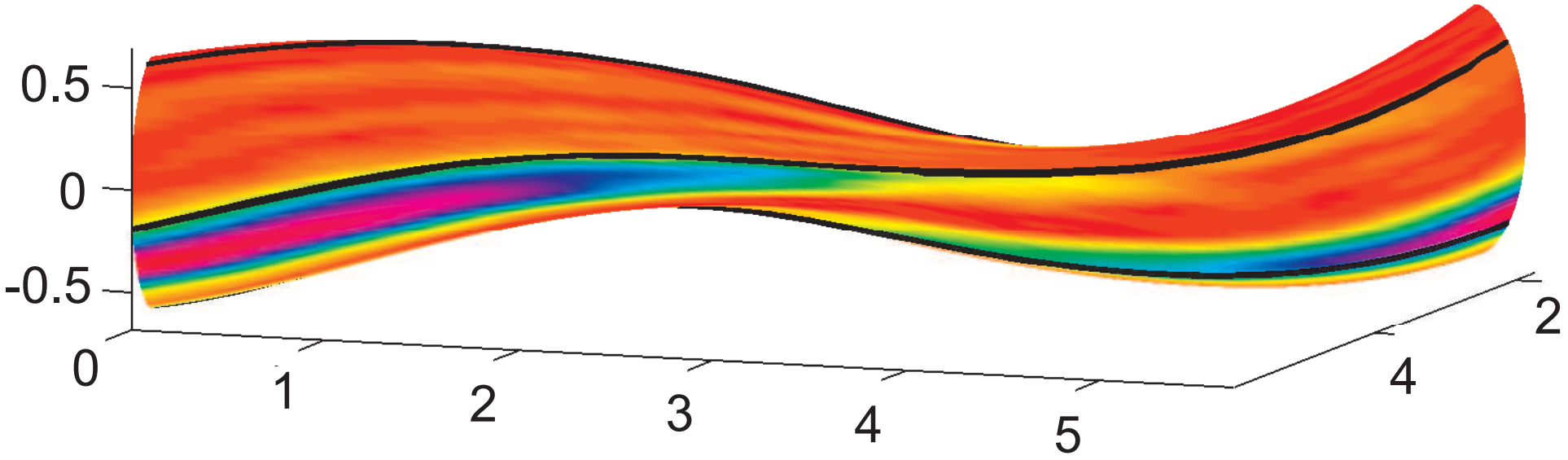
$$\sum_{\nu'} A_{\nu\nu'} \phi_{\nu'} = n_\nu$$

EUTERPE: Application to a Tokamak

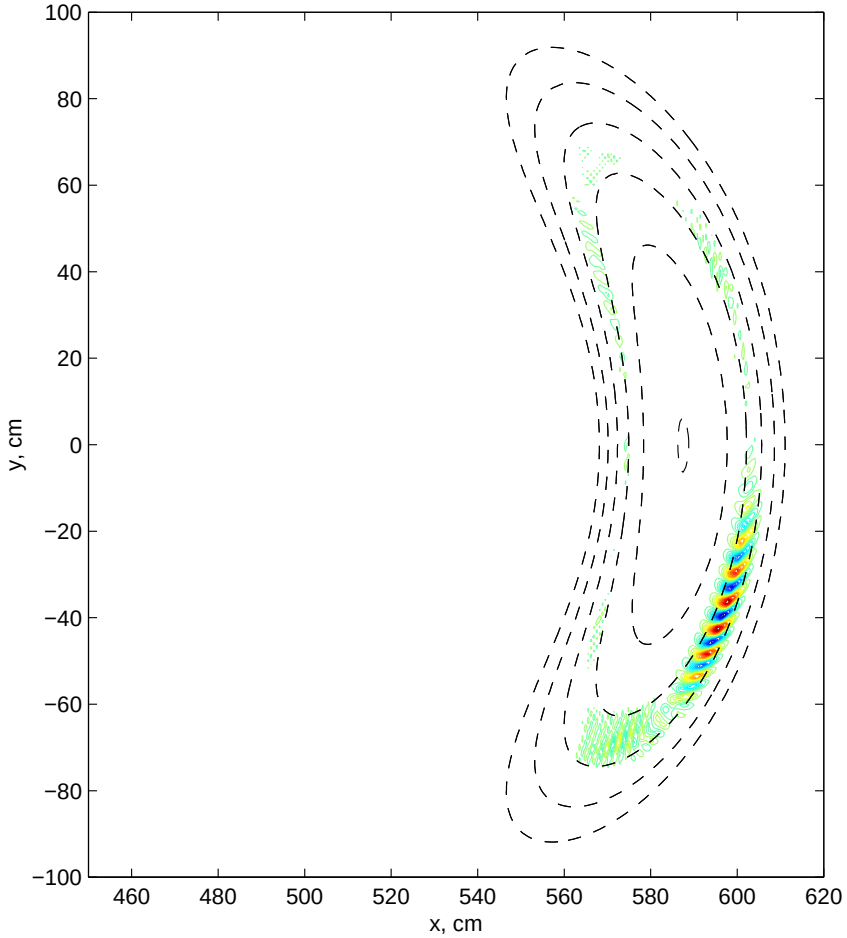


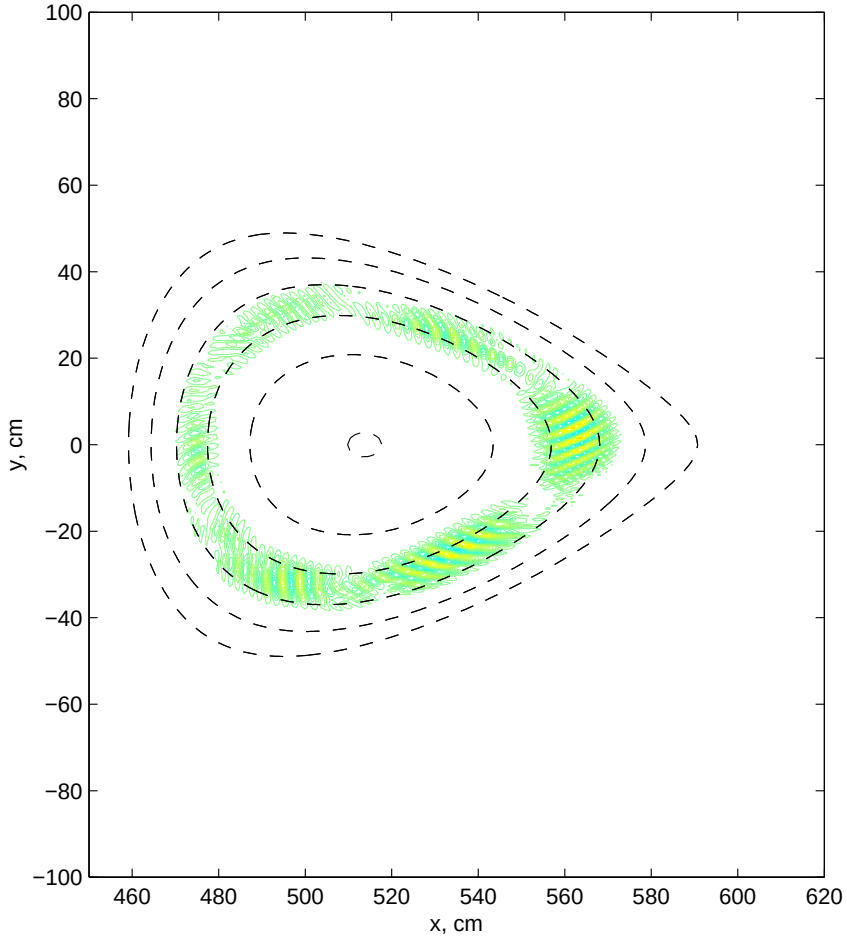


EUTERPE: Application to a Stellarator



EUTERPE: Application to a Stellarator





Summary

- Computational plasma physics combines the difficulties of theoretical plasma physics, numerical mathematics and applied computer science.
- Efficient and accurate algorithms are of paramount importance
- When writing code, make sure you comply with the language standard. Port your code!
- No code is trusted until it has been benchmarked against other codes
  $\Rightarrow$ cooperation between institutes.

The Future

- Tokamak and stellarator scientists are beginning to join forces.
- Many problems have only been investigated in simplified and often local geometries
- Integrated modelling of fusion devices is appearing on the horizon.
- The future belongs to 3d codes on parallel machines.