2D ϕ^4 theory: A study in OpenACC with PGI++

Dean Howarth

June 8, 2020

Table of contents

Coming up...

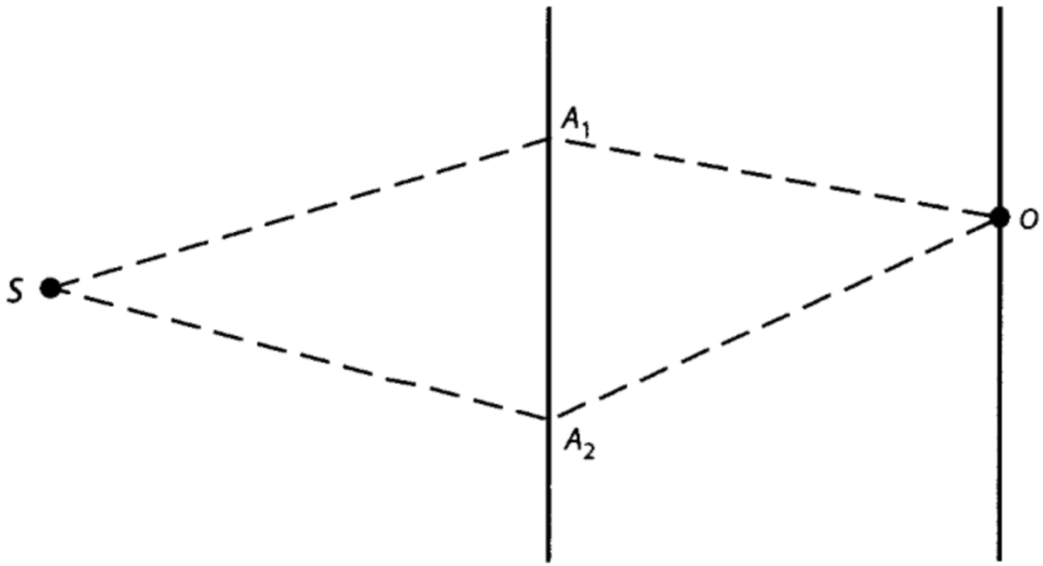
1. We need a concrete physical model to use: 2D ϕ^4
2. Within the 2D ϕ^4 code are some important algorithms (HMC, cluster decomposition)
3. Identify opportunities for parallelism
4. Use OpenACC and PGI++ to exploit those opportunities.
5. Inspect our optimisations using profiling tools, gain insights into further progress
6. Apply all of the above to your code.

Why study 2D ϕ^4 theory?

1. We know all the answers! Same universality class as 2D Ising model
2. Contains most of the ingredients required for computational physics problems
   1. Boundary conditions (periodic)
   2. A differential operator
   3. Markov chain dynamics
3. We don't actually know all the answers... (Google "2D ϕ^4 " for recent results)
   1. Long range coupling still an ongoing debate
   2. Add one more dimension, you're in unknown territory

Path Integrals: A lightning tour

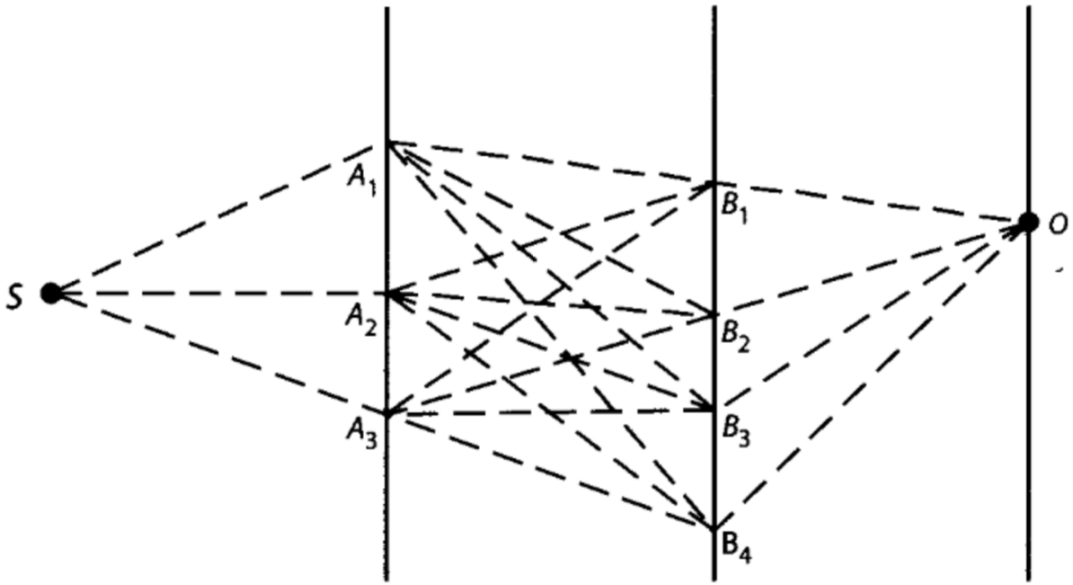
Double slit experiment..



$$P(S \to O) = |\psi_{A1} + \psi_{A2}|^2 = |\psi_{A1}|^2 + |\psi_{A2}|^2 + 2|\psi_{A1}||\psi_{A2}| \cdot (\text{phase})$$

Path Integrals: A lightning tour

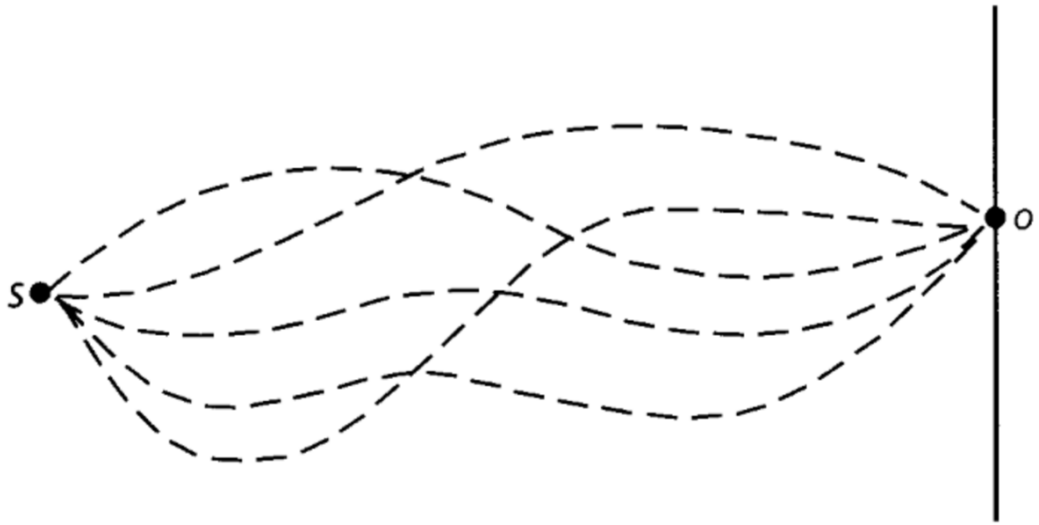
Multiple slits $(n, m, l \dots)$ in multiple panels $(A, B, C, \dots)$.



$$P(S \to O) = |(\psi_{A1} + \cdots + \psi_{An}) \cdot (\psi_{B1} + \cdots + \psi_{Bm}) \dots|^2 = \dots$$

Path Integrals: A lightning tour

Let the number of slits and panels go to infinity!



$P(S \to O) = \ldots$ we need a path integral formulation.

Path Integrals: A lightning tour

Each path contributes an amplitude (weight) to the final answer. The further away from the classical path, the less weight is given to the path. These weights are combined into 'path integral'

$$\langle \phi^2 \rangle = \frac{\int D\phi exp(iS[\phi])\phi^2}{\int D\phi exp(iS[\phi])}$$

where

$$S(\phi) = \int dx^2 \mathcal{L}(x)$$

$$\mathcal{L}(x) = \frac{1}{2}(\nabla\phi(x))^2 - \frac{1}{2}m^2\phi^2(x) - \frac{1}{4!}\lambda\phi^4(x) \quad x = (x, t)$$

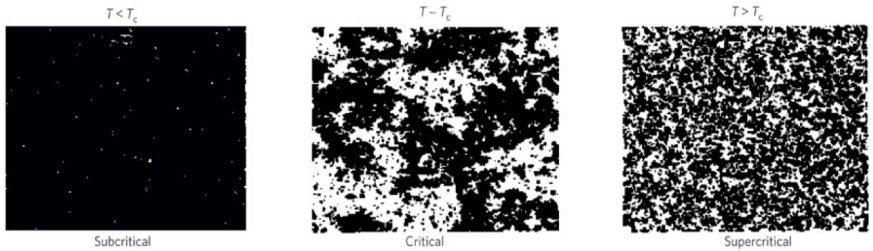
Universality Class

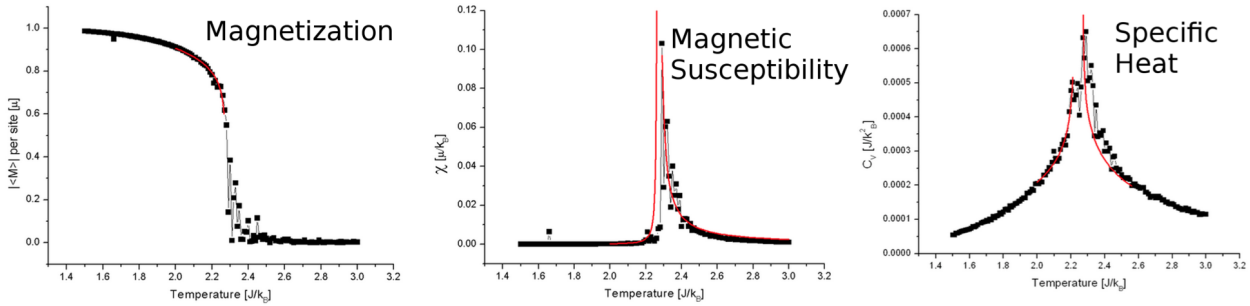
2D ϕ^4 theory and the 2D Ising Model are in the same universality class. This means that under the process of renormalisation group flow, the are exactly the same model.
What does this mean?

- At the small (local) scale, the models can be wildly different. At the infinite scale, they are indistinguishable.
- They share the same critical exponents.
- At criticality, the model loses all notions of distance. We say that the correlation length goes to infinity.

Criticality

Phase changes with the Ising model





Criticality

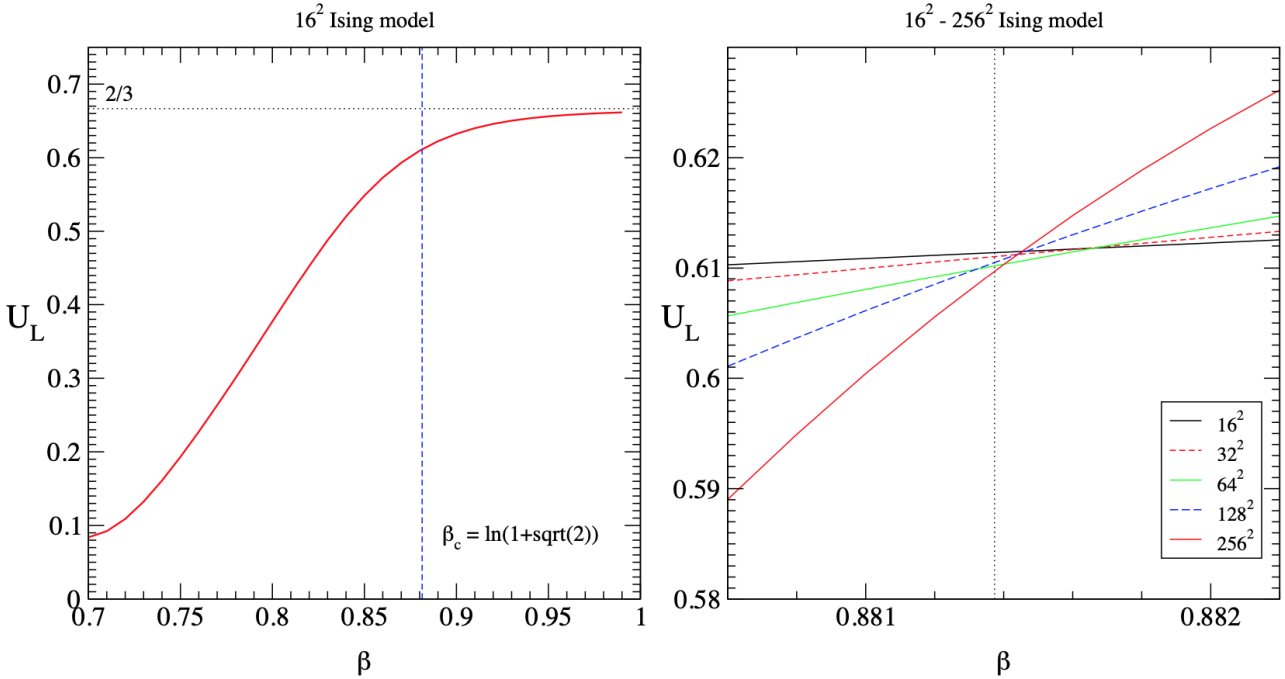
- There are two phases in this universality class: Ordered, and Disordered. Between the two phases, it is critical.
- Special types of quantum field theories have no scale: conformal field theories. The conformal field theory governing this class known as the (4,3) minimal model.
- The precise nature of the conformal field theory governing the 3D model is not known. Conformal Bootstrap methods (as well as Monte Carlo methods) are narrowing the parameter space.

Observables

- All of the physical observables are derived from 'moments' of the distribution of ϕ: $\langle\phi\rangle$, $\langle\phi^2\rangle$, and $\langle\phi^4\rangle$, with $|\phi|$ thrown in for good measure.
- An observable of particular interest is the Binder cumulant: $1.0 - \frac{\langle\phi^4\rangle}{3.0\langle\phi^2\rangle^2}$

Binder Cumulant: Kari Rummukainen's notes

Example: 2d Ising model



Left: U_L measured from 16^2 simulation. Right: volumes $16^2 - 256^2$, zooming in very close to the critical point β_c. Clearly, the slope of U_L becomes *larger* as L increases. The intersection points move only slightly.

Binder Cumulant: Kari Rummukainen's notes

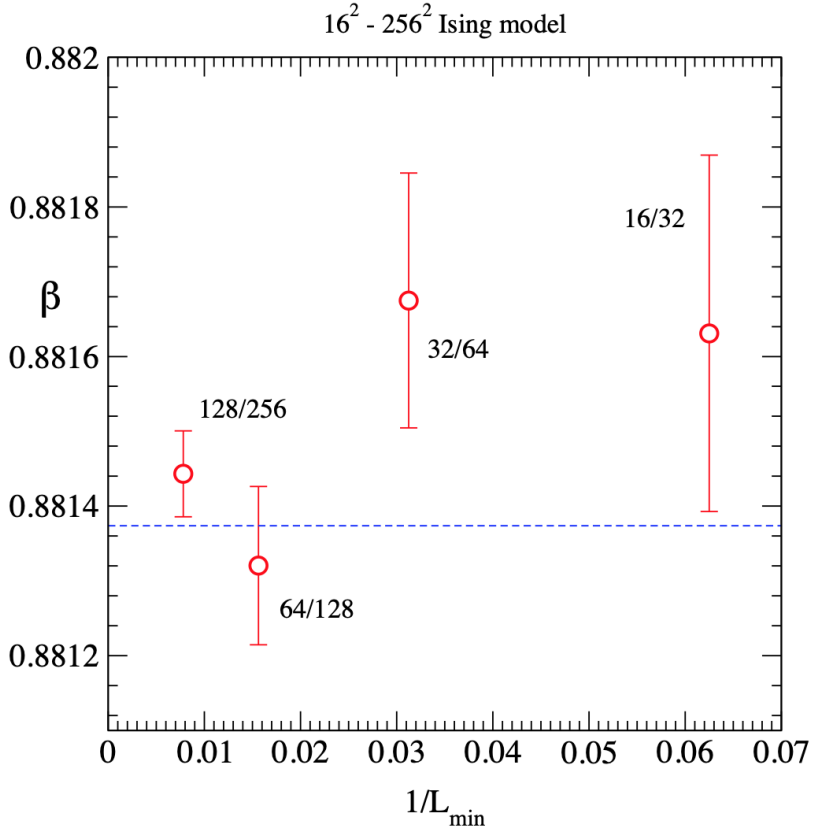
How the intersection point evolves as L increases? There is no clear finite L systematics (within the statistical errors) at various L. The largest volume pair $128/256$ alone gives

$$\beta_c^{128/256} = 0.88144(6),$$

which is already practically compatible with the known value

$$\beta_c \approx 0.88137$$

(Compare this to the maximum of $\chi_{|M|}$!)



Markov Chains

What happens when one can't solve the path integral? How does one simulate a quantum field theory? Probabilistically.

Quantum fields are superpositions of classical field configurations. If we can identify a representative sample of those configurations, we will be able to to compute a good estimate of an expectation value.
This is in perfect analogy with statistical sampling: a robust estimate of the mean of a data set can be obtained via an unbiased sample from that data set.

Markov chains offer us the ability to create such a sample. But first, we must discretise.

Discretising the Lagrangian

The continuum 2D ϕ^4 **Minkowski** Lagrangian

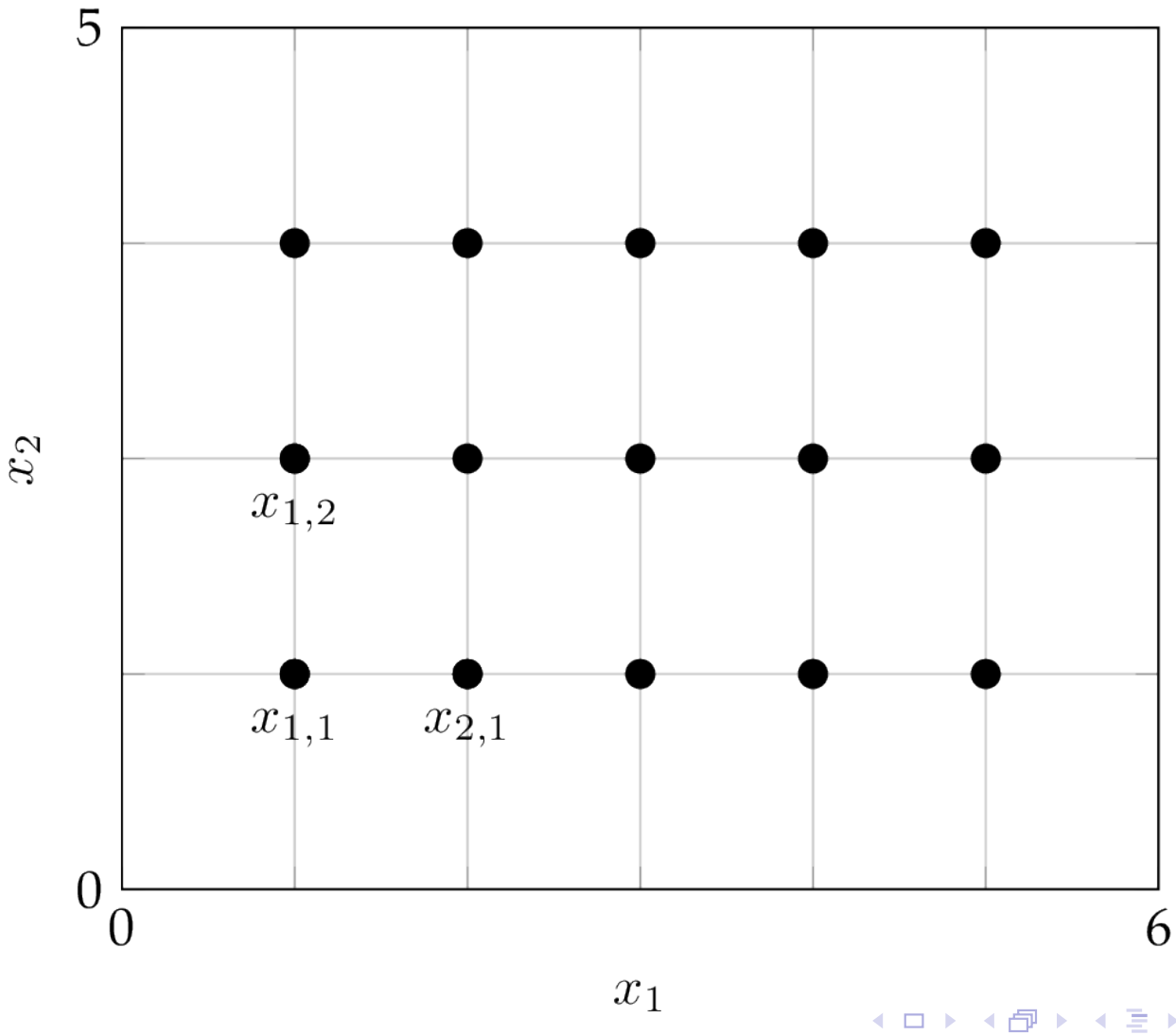
$$\mathcal{L}(x) = \frac{1}{2}(\nabla\phi(x))^2 - \frac{1}{2}m^2\phi^2(x) - \frac{1}{4!}\lambda\phi^4(x) \quad x = (x,t) \qquad (1)$$

The discrete 2D ϕ^4 **Euclidean** Lagrangian

$$\begin{aligned}\mathcal{L}_E(x) =& \frac{1}{2}[(\phi(x + an_x) - \phi(x))^2 + (\phi(x + an_y) - \phi(x))^2] \qquad (2)\\ &+ \frac{1}{2}m^2\phi^2(x) + \frac{1}{4!}\lambda\phi^4(x) \quad x = (x,y)\end{aligned}$$

Discretisation of the differential operator introduces errors. Can you work out what those errors are? (see me during the break)

Discretising the Lagrangian



Hybrid Monte-Carlo and Importance sampling

If you Google "Hybrid Monte-Carlo and Importance sampling" you will be occupied for weeks! For now, you only need to understand the concepts.

- Wick Rotation: rotate the t-dimension to $-it$

$$\langle \phi^2 \rangle_E = \frac{\int D\phi exp(-S_E[\phi])\phi^2}{\int D\phi exp(-S_E[\phi])}$$

- Fictitious momenta: noise with zero mean

$$\langle \phi^2 \rangle_E = \frac{\int D\phi exp(-\frac{1}{2}\pi^2 - S_E[\phi])\phi^2}{\int D\phi exp(-\frac{1}{2}\pi^2 - S_E[\phi])}$$

- Fictitious hamiltonian: field evolution through an extra dimension

$$\mathcal{H}_E(P, \phi) = \frac{1}{2}\pi^2 + S_E[\phi]$$

Hybrid Monte-Carlo and Importance sampling

We evolve the ϕ field through a fictitious time variable τ using the Hamiltonian equations of motion.

$$\frac{\partial \phi}{\partial \tau} = \frac{\partial \mathcal{H}}{\partial \pi}, \quad \frac{\partial \pi}{\partial \tau} = -\frac{\partial \mathcal{H}}{\partial \phi} = -\frac{\partial S}{\partial \phi} \tag{3}$$

We can solve these equations (evolve the ϕ field) using Leapfrog algorithm (See phi4Serial.cpp and Stefan's notes). We can explore the phase space further using cluster algorithms. (See WolffCluster.pdf for more details)

The HMC algorithm for *phi*^{4} theory

Here is the HMC, with annotations (search for 'Step n:') directing you to the relevant code in phi4Serial.cpp

- Step1: choose gaussian distributed momenta $P_i \in exp(-\pi_i^2/2)$
- Step2: evolve the ϕ and π fields to candidate configurations ϕ' and π':

$$\frac{d}{d\tau}\phi_x(\tau) = \frac{\partial}{\partial \pi_x}\mathcal{H}(\pi(\tau), \phi(\tau))$$
$$\frac{d}{d\tau}\pi_x(\tau) = -\frac{\partial}{\partial \phi_x}\mathcal{H}(\pi(\tau), \phi(\tau))$$

- Step3: accept candidate configurations with probability

$$P_{accept} = min(1, \exp(-(\mathcal{H}' - \mathcal{H})))$$

The HMC algorithm for *phi*^{4} theory

Step 2 in a bit more detail. (search for 'Step 2.n:')

- Step 2.1: $\pi_{1/2} = \pi_0 - \frac{d\tau}{2} F(\phi)$
- Step 2.2: For $k = 1...n-1$

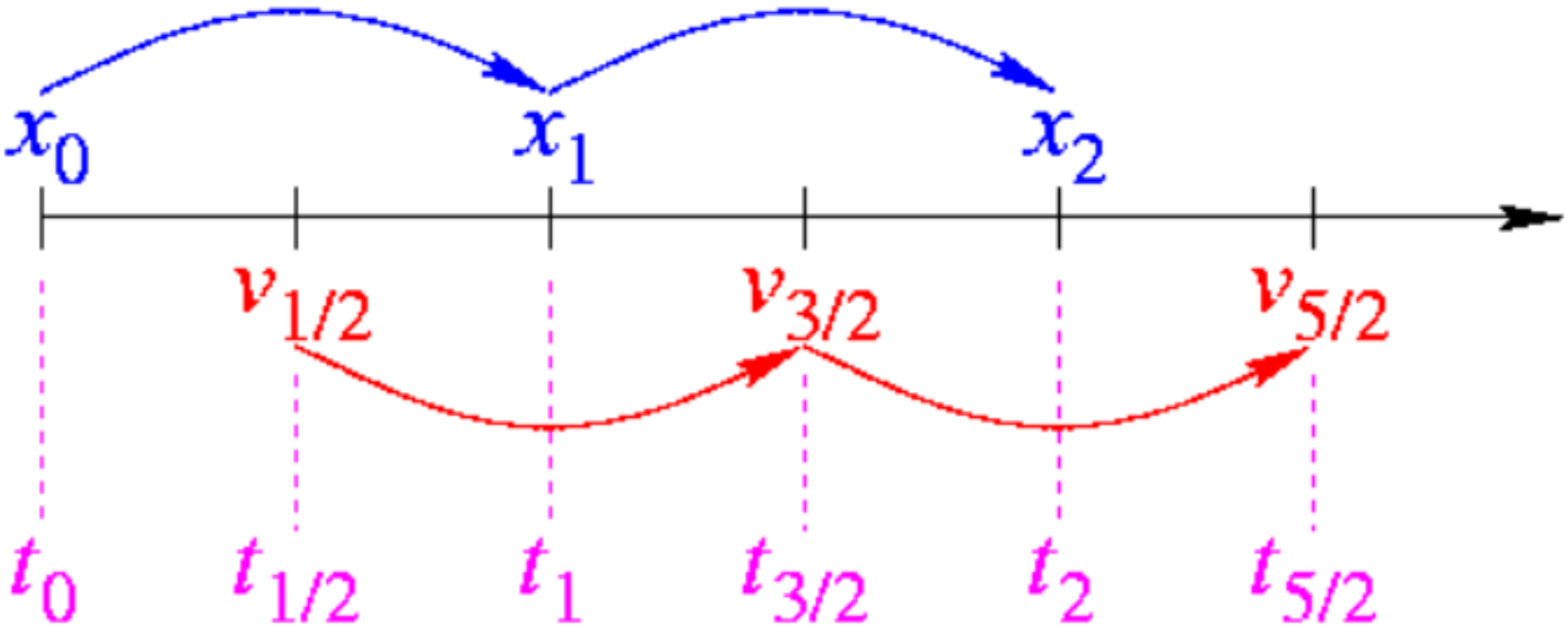
$$\phi_k = \phi_{k-1} + \pi_{k-1/2} d\tau$$
$$\pi_{k+\frac{1}{2}} = \pi_{k-\frac{1}{2}} - F(\phi) d\tau$$

- Step 2.3: $\phi_n = \phi_{n-1} + \pi_{n-\frac{1}{2}} d\tau$

$F(\phi) = \nabla S(\phi)$ is a fictitious force, driven by the 'action potential'.

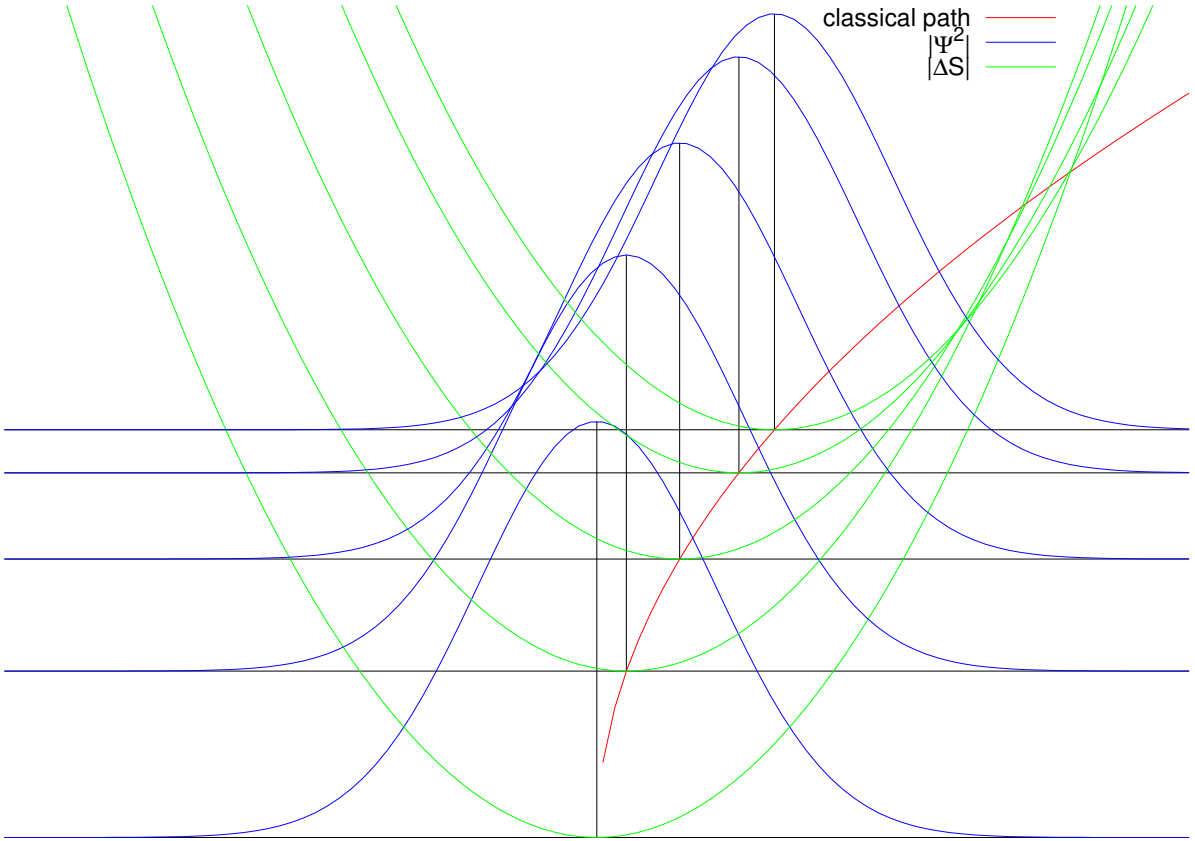
The HMC algorithm for *phi*^{4} theory

This is why the velocity Verlet algorithm is often called Leapfrog...



The HMC algorithm for *phi*^{4} theory

This is why the we call it the 'action potential'...



Cluster Algorithms

Cluster decomposition of ϕ^4 theory can lead to some staggeringly efficient evolution algorithms. It was pioneered by Fortuin and Kasteleyn in the late 60s. Omitting the gory details, one can write the partition function of a lattice theory as the sum of all possible clusters. One constructs such clusters with a probability,

$$P[bond(x, y)] = 1 - \exp(-2\phi(x)\phi(y)) \tag{4}$$

Swendsen and Wang used this result to produce an algorithm that, without rejection, moves a lattice from one configuration to another. Ulli Wolff came up with one too.

Swendsen-Wang And Wolff

Swendsen-Wang:

- Step 1: Test every n.n. bond ONCE to see if they are connected via a bond using $P[bond(x,y)] = 1 - \exp(-2\phi(x)\phi(y))$
- Step 2: Identify every connected cluster on the lattice.
- Step 3: Flip each cluster with 50% probability.

Wolff:

- Step 1: Choose a lattice site at random.
- Step 2: Identify every site connected to that original site using $P[bond(x,y)] = 1 - \exp(-2\phi(x)\phi(y))$
- Step 3: Flip the cluster with 100% probability.

You have been provided with notes on the Wolff algorithm, and code for the Swendsen-Wang. Can you write a Wolff cluster algorithm in the activity time?

The serial implementation

Provided under
RPI-Advanced-Cyberinfrastructure/phi4AndOpenACC/phi4Serial
is a set of routines that perform HMC on a ϕ^4 theory. Let's read through the code and understand it before using PGI++.
Points to consider:

- HMC is performed once for multiple cluster updates... why?
- Can you decipher what the cluster algorithm does?
- The trajectory() function performs the Leapfrog integration. Can you see how it does it?
- Looking ahead, can you intuitively see where the routines can be easily parallelised?

What are OpenACC and PGI++?

PGI++ is an OpenACC compiler. It is able to compile C++ standard code, along with OpenACC compliant pragmas:

```
// Copy lattice field (Device)
template<typename T> inline void copyField(T phi2[L][L], T phi1[L][L]) {
  {
#pragma acc parallel loop collapse(2)
    for(int x=0; x<L;x++)
      for(int y=0; y<L;y++)
        phi2[x][y] = phi1[x][y];
  }
}

// Copy lattice field (Host)
template<typename T> inline void copyField_host(T phi2[L][L], T phi1[L][L]) {

  for(int x=0; x<L;x++)
    for(int y=0; y<L;y++)
      phi2[x][y] = phi1[x][y];
}
```

A pragma is a 'hint' or 'directive' to the compiler. In the above excerpt, we have informed the compiler that the (2) nested loop may be collapsed. I.e., each iteration is independent.

OpenACC Vs OpenMP

- OpenMP is a large collection of specific directives to the compiler. You can make a program highly optimal for one architecture, but it could be hopelessly poor on another.
- OpenACC is much leaner. The compiler makes a lot of decisions "under the hood" and it's up to the programmer to make sure there are no race conditions.
- This lecture is about making simple code go faster without having to study hardware architecture diagrams. OpenACC gets us to where we want to be in quick order.
- C++ (17) is moving more towards accepting parallelism. The new pSTL library allows programmers to express parallelism in their STL calls, E.g., sum, copy. Code written in the future will have parallel directives built in: The user won't even know their code was parallelised!

Some Notes on GPU parallelism Vs CPU parallelsim

- CPU parallelism is straightforward:
  - Do more things with more CPUs!
  - No issues with moving data around, but usually there are only O(100) cores on a compute node.
  - Very few situations where a parallel CPU application doesn't give speed up.
- GPU parallelism is more involved:
  - Device memory constraints.
  - GPU bandwidth constraints
  - Shared memory / block and thread geometry
  - Sometimes GPU parallelising isn't worth the overhead.

Using OpenACC and PGI++

We are going to learn OpenACC by example. Having seen and understood some serial code, we shall add OpenACC pragmas where relevant and explain each addition in context.

- v0: The serial code, with timings.
- v1: Lots of OpenACC additions to the HMC.
- v2: More OpenACC in the cluster decomposition.

  ACTIVITY CHALLENGES
- v3: Can you write a Wolff algorithm? Can you make the HMC more efficient? Can you write a two-point correlation function measurement $\langle\phi(x)\phi(y)\rangle$ and make it run at nosebleed speed? Can you implement a GPU based RNG?

Ask questions, work in groups, read the code comments, happy hacking!