Abstract Representations for LQCD

Université de Versailles St Quentin/INRIA

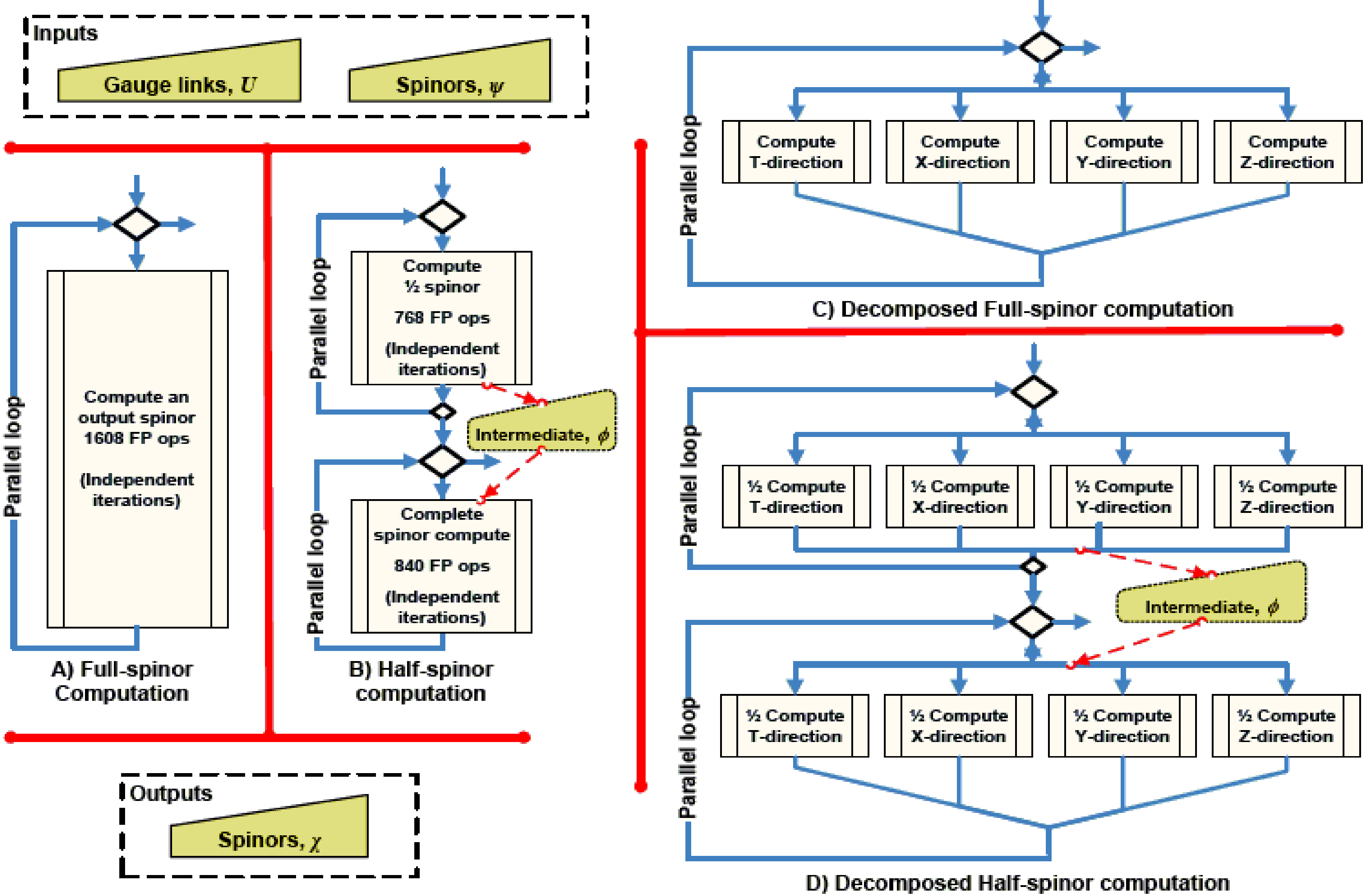
D. Barthou

LQCD ETMC Application

Current ETMC code

- C code, hand optimized for BG and SSE,
- Hopping_Matrix function represents hot spot,
- complex data structures, 4D torus lattice, stencil computation

LQCD ETMC Application



LQCD ETMC Application

Current ETMC code

- C code, hand optimized for BG and SSE,
- Hopping_Matrix function represents hot spot,
- complex data structures, 4D torus lattice, stencil computation

Optimizations performed in projects PARA

- Very different from one architecture to the other
  - SIMDization, tiling, loop spliting, unroll, ...
  - Most of it by hand, limited to Hopping_Matrix
- *Performance limited by mem. bandwidth* (IA64, Cell)
- Still lacking 1 or 2 orders of magnitude in perf. for Petaflop...

Improving performance?

Necessity to improve computation/memory access ratio, or bandwidth

- **Change architecture**
  - Change bandwidth (*wait for André's talk*)
  - Need to adapt code (SIMD, tile sizes, unrolling factors, ...)
- **Improve data reuse (reduces memory accesses)**
  - Existing reuse of Hopping_Matrix
  - Some data structure reused many times without modification between calls to HM (gauge links)
  - Same data modified multiple times accross calls to HM (tiling time). Benefits ?

Why a higher abstraction?

One representation for all architectures

- Code generators needed

Widen space of possible transformations

- Richer semantics
- Control/data structures can be adapted to architecture

1) maths/physics
   - no schedule, not executable
2) dataflow or domain-specific language
   - schedule not fully specified, parallelism not explicit
3) source code
   - full schedule, data structure optimized
   - different levels (C, asm, binary at runtime,...)

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Some higher abstractions

1) maths/physics
   - SPIRAL, Fortress with efforts
2) dataflow or domain-specific language
   - TAO (apeNEXT),
   - sigma-SPL (spiral),
   - Fortress+libraries (Sun),
   - Dataflow rep. (Systems of Affine recurrent equations for instance)

Tao Language

Toolkit for Advanced Optimization

Features

- Domain specific language, based on libraries for large scale optimization problems
- Linear solvers, manipulation of matrices/vectors
- Parallelism is inside libraries
- Possible to mix C code with library calls

Fortress

Features

- Write code as maths, for scientific computing
  - type inference for guessing operators described by a blank
- User-defined iterators
  - iterators subdivide a space with sub-iterators
  - implicitely parallel (sequential must be specified)
  - architecture specific
- Transactional memory, atomic instruction blocks
- No compiler (not yet), interpreted. OpenSource
  - JVM

Fortress: SUN example

```
conjGrad(A: Matrix[\Float\], x: Vector[\Float\]):
        (Vector[\Float\], Float)
  cgit_max = 25
  z: Vector[\Float\] := 0
  r: Vector[\Float\] := x
  p: Vector[\Float\] := r
  rho: Float := r^T r
  for j <- seq(1:cgit_max) do
    q = A p
    alpha = rho / p^T q
    z := z + alpha p
    r := r - alpha q
    rho0 = rho
    rho := r^T r
    beta = rho / rho0
    p := r + beta p
  end
  (z, ||x - A z||)


(z,norm) = conjGrad(A,x)
```

Matrix[\T\] and Vector[\T\] are parameterized interfaces, where T is the type of the elements.

Fortress: SUN example

```
conjGrad⟦Elt extends Number, nat N,
         Mat extends Matrix⟦Elt,N×N⟧,
         Vec extends Vector
        ⟧(A: Mat, x: Vec): (Vec, Elt)
  cgit_max = 25
  z: Vec := 0
  r: Vec := x
  p: Vec := r
  ρ: Elt := r^T r
  for j ← seq(1:cgit_max) do
     q = A p
     α = ρ / p^T q
     z := z + α p
     r := r - α q
     ρ0 = ρ
     ρ := r^T r
     β = ρ / ρ0
     p := r + β p
  end
  (z, ‖x - A z‖)
```

This would be considered entirely equivalent to the previous version. You might think of this as an abbre-viated form of the ASCII version, or you might think of the ASCII version as a way to conveniently enter this version on a standard keyboard.

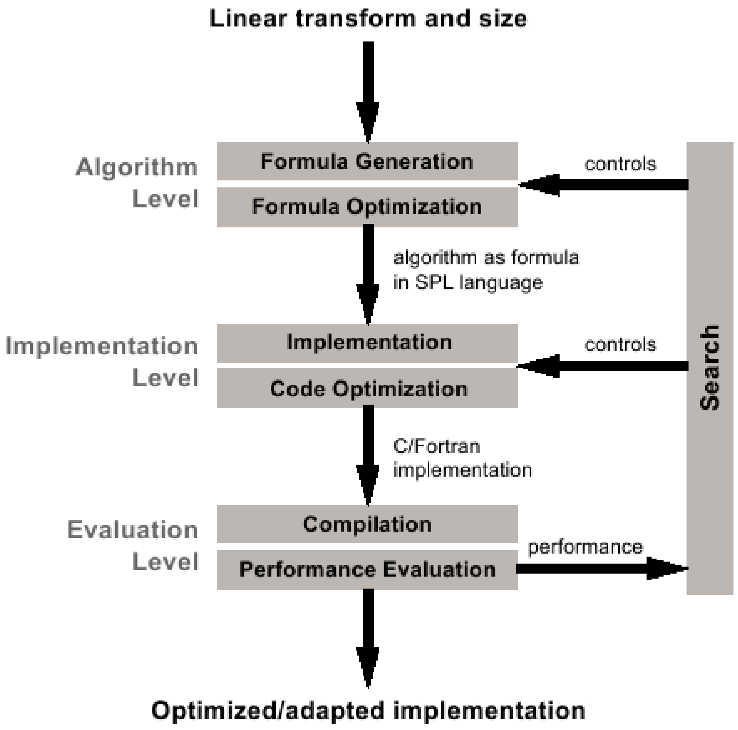
Spiral

Features

- Domain-specific language for DFTs, DCTs, linear algebra and other signal processing functions
- One-line mathematical formula
  - matrix product, tensor product, direct sum
- *Based on divide & conquer breakdown rules*
  - decompose a formula into simpler ones
  - multiple breakdown rules for different decomposition algorithms/variants
- From formula to optimized code
  - code generation for Cell, GPU, BG, multicores...
  - not yet inter-node parallelism

Spiral code generation

- Based on search
  - different formulations
  - code versions
- Many back ends
  - BG, Cell, GPU, multicores
- High level of performance



Spiral: breakdown rules

$$\mathbf{DFT}_n \longrightarrow (\mathbf{DFT}_k \otimes I_m) D_{k,m} (I_k \otimes \mathbf{DFT}_m) L^n_k \tag{2.1}$$

$$\mathbf{DFT}_n \longrightarrow V^{-1}_{m,k} (\mathbf{DFT}_k \otimes I_m)(I_k \otimes \mathbf{DFT}_m) V_{m,k} \tag{2.2}$$

$$\mathbf{DFT}_n \longrightarrow W^{-1}_n (I_1 \oplus \mathbf{DFT}_{n-1}) E_n (I_1 \oplus \mathbf{DFT}_{n-1}) W_n \tag{2.3}$$

$$\mathbf{DFT}_n \longrightarrow B^\top_{n,m} D_m \, \mathbf{DFT}_m \, D'_m \, \mathbf{DFT}_m \, D''_m B_{n,m}, \quad m \geq 2n-1 \tag{2.4}$$

$$\mathbf{DFT}_n \longrightarrow P^\top_{k/2,2m} \left(\mathbf{DFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i C_{2m} \, \mathbf{rDFT}_{2m}((i+1)/k)\right)\right) \left(\mathbf{RDFT}'_k \otimes I_m\right) \tag{2.5}$$

$$\begin{vmatrix} \mathbf{RDFT}_n \\ \mathbf{RDFT}'_n \\ \mathbf{DHT}_n \\ \mathbf{DHT}'_n \end{vmatrix} \longrightarrow (P^\top_{k/2,m} \otimes I_2) \left(\begin{vmatrix} \mathbf{RDFT}_{2m} \\ \mathbf{RDFT}'_{2m} \\ \mathbf{DHT}_{2m} \\ \mathbf{DHT}'_{2m} \end{vmatrix} \oplus \left(I_{k/2-1} \otimes_i M_{2m} \begin{vmatrix} \mathbf{rDFT}_{2m}((i+1)/k) \\ \mathbf{rDFT}_{2m}((i+1)/k) \\ \mathbf{rDHT}_{2m}((i+1)/k) \\ \mathbf{rDHT}_{2m}((i+1)/k) \end{vmatrix} \right) \right) \cdot \left(\begin{vmatrix} \mathbf{RDFT}'_k \\ \mathbf{RDFT}'_k \\ \mathbf{DHT}'_k \\ \mathbf{DHT}'_k \end{vmatrix} \otimes I_m \right) \tag{2.6}$$

$$\mathbf{RDFT}_n \longrightarrow D_n \cdot \mathbf{DCT\text{-}2}_n \cdot P_n, \quad n \text{ odd} \tag{2.7}$$

$$\mathbf{DCT\text{-}2}_n \longrightarrow P^\top_{k/2,2m} \left(\mathbf{DCT\text{-}2}_{2m} K^{2m}_2 \oplus \left(I_{k/2-1} \otimes N_{2m} \, \mathbf{RDFT\text{-}3}^\top_{2m}\right)\right) G_n (L^{n/2}_{k/2} \otimes I_2) \cdot (I_m \otimes \mathbf{RDFT}'_k) Q_{m/2,k} \tag{2.8}$$

$$\mathbf{DCT\text{-}2}_n \longrightarrow L^n_{n/2} \cdot (\mathbf{DCT\text{-}2}_{n/2} \oplus \mathbf{DCT\text{-}4}_{n/2}) \cdot \begin{bmatrix} I_{n/2} & J_{n/2} \\ \end{bmatrix} \tag{2.9}$$

Dataflow representations

Dataflow only represents flow of values

Features

- Multiple schedules possible (seq or //)
- Different languages/formalizations
  - Lustre, StreamIT, Khan networks,
  - or just systems of affine recurrence equations
- Choice of efficient data structures and parallelism can be derived from initial form
- Scheduling/transformations/code generation for polyhedral model applies when control and access patterns are regular

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Conclusions

- Library based representation (TAO)
  - Wraps computation inside functions, compact representation.
- Fortress:
  - No compiler, just facilitating code representation. Optimistic view: makes a formula executable.
- Spiral:
  - Need to simplify LQCD computation for Spiral, efficient code generation
- Dataflow representation:
  - Focusing on dependences, independent slices, no impact on the writing of the operations. Possible to detect vector operations/SIMD.
- **Others to be explored / used ?**

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES